O'REILLY



图灵程序设计丛书

命令行中的数据科学

DATA SCIENCE AT THE COMMAND LINE

[荷] Jeroen Janssens 著

王晓伟 刘峰 译



中国工信出版集团



人民邮电出版社

POSTS & TELECOM PRESS

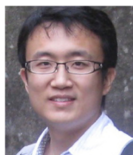
数字版权声明

译者介绍



王晓伟

毕业于国防科学技术大学，获计算机科学与技术专业博士学位。研究兴趣为海量数据管理与挖掘。



刘峰

百度LBS位置大数据部资深研发工程师，新加坡南洋理工大学计算机工程系博士，研究领域包括机器学习、神经网络、数据挖掘等。2010年加入百度，主要从事大数据分析和挖掘方面的工作，近年来专注于网络定位、智能交通等LBS大数据的挖掘和机器学习应用。

 图灵程序设计丛书

命令行中的数据科学

Data Science at the Command Line

Facing the Future with Time-Tested Tools

［荷］Jeroen Janssens 著

王晓伟　刘峰 译

人 民 邮 电 出 版 社

北　　京

图书在版编目（CIP）数据

命令行中的数据科学 /（荷）詹森斯（Janssens, J.）著；王晓伟，刘峰译. -- 北京：人民邮电出版社，2015.6
（图灵程序设计丛书）
ISBN 978-7-115-39168-1

Ⅰ. ①命… Ⅱ. ①詹… ②王… ③刘… Ⅲ. ①数据处理 Ⅳ. ①TP274

中国版本图书馆CIP数据核字(2015)第087967号

内 容 提 要

本书集实用性和先进性于一身，为数据分析人员使用命令行这个灵活的工具提供了重要参考。作者讲解了众多实用的命令行工具，以及如何使用它们高效地获取、清洗、探索和建模数据。无论你使用 Windows、OS X，还是 Linux，都可以安装包含 80 多个命令行工具的“数据科学工具箱”，迅速建立自己的数据分析环境。无论你是否已经习惯于使用 Python 或 R 语言，都能够通过本书体会到使用命令行的快捷、灵活与伸缩自如。

本书适合各层次的软件开发人员，包括专业和非专业的数据分析人员。

◆ 著 [荷] Jeroen Janssens
译 王晓伟 刘 峰
责任编辑 岳新欣
责任印制 杨林杰

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京 印刷

◆ 开本：800×1000 1/16
印张：11.75
字数：242千字 2015年 6 月第 1 版
印数：1－3 500册 2015年 6 月北京第 1 次印刷

著作权合同登记号 图字：01-2015-2578号

定价：49.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

版权声明

O'Reilly Media, Inc.介绍

O'Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O'Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些"细微的信号"来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O'Reilly 为软件开发人员带来革命性的"动物书"；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项 O'Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

"O'Reilly Radar 博客有口皆碑。"

——*Wired*

"O'Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。"

——*Business 2.0*

"O'Reilly Conference 是聚集关键思想领袖的绝对典范。"

——*CRN*

"一本 O'Reilly 的书就代表一个有用、有前途、需要学习的主题。"

——*Irish Times*

"Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：'如果你在路上遇到岔路口，走小路（岔路）。'回顾过去 Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。"

——*Linux Journal*

献给我的妻子 Esther。如果没有她的鼓励、支持和耐心，我不可能完成这本书。

目录

前言

数据科学是个激动人心却又非常年轻的领域。不幸的是，许多个人和公司，总是认为需要利用新技术才能解决数据科学提出的问题。实际上，正如本书所揭示的，许多问题使用命令行就能解决，而且有时候效率要高得多。

大约 5 年前，在攻读博士学位期间，我逐步从使用微软 Windows 转为使用 GUN/Linux。刚开始我有点谨小慎微，因此同时安装了这两个操作系统（也就是双系统启动）。后来，在这两个系统之间切换的需求越来越少，有时我甚至对 Arch Linux 修修补补，能从零开始自己定制操作系统。这时能用的只有命令行，而且想做什么完全随心所欲。很快，我就对使用命令行得心应手。最终，由于业余时间越来越宝贵，我决定使用名为 Ubuntu 的 GNU/Linux 发行版，因为它易于使用并且有庞大的社区。尽管如此，命令行仍然是我完成绝大部分工作的不二选择。

实际上，我后来认识到，命令行不单可以用于安装软件、配置系统以及搜索文件。于是我开始学习诸如 `cut`、`sort` 和 `sed` 这些命令行工具。这些工具都是将数据作为输入，对数据进行处理，然后打印结果。Ubuntu 自带了相当多这样的工具。当明白可以将这些小工具结合起来使用时，我就对它入迷了。

当我拿到博士学位，成为一名数据科学家时，我想充分利用这种方法来做数据科学工作。幸亏有几个新的开源命令行工具，包括 `scrape`、`jq` 和 `json2csv`，我甚至能够使用命令行来完成抓取网站以及处理大量 JSON 数据这样的任务。2013 年 9 月，我写了一篇名为“数据科学的 7 个命令行工具”的博客文章（http://jeroenjanssens.com/2013/09/19/seven-command-line-tools-for-data-science.html）。让我吃惊的是，这篇文章获得很大反响。后来许多人向我推荐其他命令行工具，于是我开始考虑是否可以将这篇文章扩充成书。令人高兴的是，10 个月之后，在许多才华横溢的人的帮助下（参见“致谢”），本书得以付梓。

分享这段个人经历不仅是想介绍本书的由来，更是希望你知道我也是需要学习命令行的。使用命令行与使用图形化用户界面迥然不同，刚开始可能是令人生畏的。但是，既然我能

够学会它，你当然也没问题。不管你目前使用的是什么操作系统，也不管你现在是以什么方式做数据科学的工作，读完本书，你也能够利用命令行的强大能力。即使你已经熟悉命令行，或者甚至已经打算学习 shell 脚本，你仍然可能在书中发现一些有趣技巧或命令行工具，能用于未来的数据科学项目。

从本书可以学到的

书中将对大量数据进行获取、清洗、探索以及建模。我们不会过多介绍如何完成这些数据科学任务，因为对于诸如应该何时及用什么进行统计检验，或者怎样才能将数据可视化做到最好，很容易找到大量参考资料。本书致力于实用性，旨在通过教你用命令行执行数据科学任务，使你更加高效和多产。

尽管书中讨论了 80 多个命令行工具，但这些工具本身并不是最重要的。有些命令行工具存在已久，有些则是新近出现，并且可能最终会被更好的工具所取代。甚至在你阅读本书的时候，有的命令行工具正在创建之中。在过去的 10 个月里，我就已经发现了许多奇妙的命令行工具。遗憾的是，有的工具被发现的时间太晚，无法包含在本书中。总之，命令行工具的新陈代谢是常态。

用工具、管道和数据进行工作的思想才是最重要的。多数命令行工具只做一项任务，并且做得很好。这符合 Unix 的理念，这种理念在书中许多地方都有体现。一旦熟悉了命令行，并且学会了如何将命令行工具结合起来，你就学会了一项非常宝贵的技能。如果还能创建新的工具，那你就出类拔萃了。

怎样阅读本书

一般来说，我们建议你按顺序阅读。书中介绍的概念或者命令行工具，很可能会在下一章中采用。例如在第 9 章中使用了 `parallel` 工具，这个工具在第 8 章进行了深入的讨论。

数据科学是一个宽广的领域，与许多领域都有交叉，例如程序设计、数据可视化以及机器学习。因此，本书触及了许多有趣的话题，遗憾的是无法逐一详尽讨论。在书中我随时都会给出建议的参考资料。如果只是想跟着本书的节奏，并不需要学习这些参考资料，但是如果你感兴趣，可以深入研究一下这些推荐的资料。

本书面向的读者

本书面向的读者只有一个前提条件：你与数据打交道。使用哪种编程语言或者统计计算环境无关紧要。本书会在开头解释所有必要的概念。

你的操作系统是微软 Windows、Mac OS X 还是其他形式的 Unix 系统都无关紧要。书中自

带数据科学工具箱，这是一个容易安装的虚拟环境。它使你可以运行命令行工具，并可以在与本书相同的环境下，跟着一起学习这些示例代码。你不需要浪费时间来研究如何安装所有的命令行工具以及它们依赖的环境。

书中包含一些用 Bash、Python 和 R 编写的代码，如果你有一定的编程经验会有帮助，但这绝不是必要的。

排版约定

本书使用了下列排版约定。

- *楷体*
  表示新术语。
- 等宽字体（`Constant width`）
  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键字等。
- 加粗等宽字体（**`Constant width bold`**）
  表示应该由用户输入的命令或其他文本。



这个图标表示提示或建议。



这个图标表示一般注记。



这个图标表示警告或提醒。

使用代码示例

补充材料（虚拟机、数据、脚本、定制的命令行工具等）可以从 https://github.com/jeroenjanssens/data-science-at-the-command-line 下载。

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序或文档中。除非你使用了很大一部分代码，否则无需联系我们获得许可。比如，用本书的几个代码片段写一个程序就无需获得许可，销售或分发 O'Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无需获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN。比如："*Data Science at the Command Line* by Jeroen H.M. Janssens (O'Reilly). Copyright 2015 Jeroen H.M. Janssens, 978-1-491-94785-2."

如果你觉得自己对示例代码的用法超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

Safari® Books Online



Safari Books Online（http://www.safaribooksonline.com）是应运而生的数字图书馆。它同时以图书和视频的形式出版世界顶级技术和商务作家的专业作品。技术专家、软件开发人员、Web 设计师、商务人士和创意专家等，在开展调研、解决问题、学习和认证培训时，都将 Safari Books Online 视作获取资料的首选渠道。

对于组织团体、政府机构和个人，Safari Books Online 提供各种产品组合和灵活的定价策略。用户可通过一个功能完备的数据库检索系统访问 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 以及其他几十家出版社的上千种图书、培训视频和正式出版之前的书稿。要了解 Safari Books Online 的更多信息，我们网上见。

联系我们

本书有一个专属网页，你可以在那儿找到与代码无关的勘误列表以及其他信息。本书的网站地址是：

http://datascienceatthecommandline.com/

与代码、命令行工具和虚拟机相关的勘误，请通过 GitHub 的问题追踪系统提交：

https://github.com/jeroenjanssens/data-science-at-the-command-line/issues

请把对本书的评价和问题发给出版社。

美国：

O’Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）

奥莱利技术咨询（北京）有限公司

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O’Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：

http://facebook.com/oreilly

请关注我们的 Twitter 动态：

http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：

http://www.youtube.com/oreillymedia

请关注 Jeroen 的 Twitter 动态：@jeroenhjanssens

http://twitter.com/jeroenhjanssens

致谢

首先我要感谢 Mike Dewar 和 Mike Loukides，他们相信我在 2013 年 9 月撰写的博客文章“数据科学的 7 个命令行工具”可以扩写成书。感谢 Jared Lander 邀请我在纽约开放统计编程聚会上演讲，我就是在准备那次演讲时有了撰写这篇博客文章的灵感。

我要特别感谢我的技术审阅人 Mike Dewar、Brain Eoff 和 Shanes Reustle，他们不厌其烦地阅读初稿，一丝不苟地测试所有命令，并且提出了宝贵的反馈信息。你们的努力使本书得到了极大的改善。书中若还有错误，全都是我自己的责任。

我有幸与四位优秀的编辑一起工作，他们是 Ann Spencer、Julie Steele、Marie Beaugureau 和 Matt Hacker。感谢你们的指导，我通过你们认识了 O’Reilly 很多才华横溢的朋友，他们是 Huguette Barriere、Sophia DeMartini、Dan Fauxsmith、Yasmina Greco、Rachel James、Jasmine Kwityn、Ben Lorica、Mike Loukides、Andrew Odewahn 和 Christopher Pappas，以及

许多未曾谋面的在幕后英雄。正是这些朋友的共同努力使我与 O'Reilly 的合作十分愉快。

本书讨论了 80 多个命令行工具。毋庸置疑，没有这些工具，本书根本就不可能成行。因此，我要对创建这些工具以及对其有过贡献的作者们表达十二分的谢意。遗憾的是，这些作者人数众多，无法在这里一一列举，只能在附录 A 中提及。特别感谢 Araon Crow、Jehiah Czebotar、Christopher Groskopf、Dima Kogan、Sergey Lisitsyn、Francisco J. Martin 和 Ole Tange，帮助我学习他们了不起的命令行工具。

本书大量使用了数据科学工具箱，它是一个包含书中所有命令行工具的虚拟环境。它是建立在许多巨人的肩膀之上的，有鉴于此，我要感谢在 GNU、Linux、Ubuntu、Amazon Web Services、GitHub、Packer、Ansible、Vagrant 和 VirtualBox 背后的人们，没有他们就没有数据科学工具箱。感谢 Matthew Russell 在我起初开发数据科学工具箱时给我的启发和反馈。他的著作 *Mining the Social Web*（O'Reilly，http://shop.oreilly.com/product/0636920030195.do）也提供了一个虚拟机。

特别感谢我读博士期间的导师 Eric Postma 和 Jaap van den Herik。在这 5 年间他们教我学了很多课程。尽管写书与写博士论文有很大差别，但过去的 10 个月证明了许多课程都非常有帮助。

最后，我要感谢 YPlan 的同事们，感谢我的朋友、家人，特别是我的妻子 Esther，感谢她无私的支持，以及总是在恰当的时间将我从命令行旁边拉开。

第 1 章 简介

本书介绍如何用命令行来做数据科学工作。我们的目的是通过运用命令行工具的力量，使你成为一名高效和多产的数据科学家。

首先需要解释一下书名中同时出现的两个词："数据科学"与"命令行"。一项已经有 40 多年历史[1]的老技术，对于一个只有几年历史的领域还有什么用处吗？

今天，数据科学家有海量的激动人心的技术和编程语言可供选择，Python、R、Hadoop、Julia、Pig、Hive、Spark 就是其中一些例子。你可能已经用过其中一种或好几种。既然这样，为什么仍然应该关注数据科学的命令行工具？命令行提供了哪些东西，是其他技术和编程语言所不具备的？

这些都是合乎情理的问题。第 1 章将回答这些问题。首先，给出了数据科学的一个实用定义作为本书的基础。然后，列出了命令行的 5 个重要优势。最后，通过一个现实用例来说明命令行的能力和灵活性。到本章结束时，希望我们能够让你相信，为了做数据科学，命令行确实是值得学习的。

1.1 概述

本章你将学到：

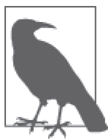
- 数据科学的一个实用定义

注 1：UNIX 操作系统的发展早在 1969 年就开始了（http://www.unix.org/what_is_unix/history_timeline.html）。从一开始，命令行就是它的重要特征，而管道这一重要概念是在 1973 年加入的。

- 命令行究竟是什么，以及如何使用命令行
- 为什么命令行是做数据科学工作的绝佳环境

1.2　数据科学就是OSEMN

数据科学领域仍然处于萌芽阶段，正因如此，对于其内涵有着各种各样的定义。在本书中，我们采用了 Mason & Winggins（2010）提出的非常实用的定义。他们按照以下 5 个步骤定义数据科学：(1) 数据获取；(2) 数据清洗；(3) 数据探索；(4) 数据建模；(5) 数据解释。这 5 个步骤一起构成了 OSEMN（发音同 awesome）模型。这个定义是本书的基础，因为每个步骤（除了步骤 5）都有对应的一章。下面用 5 个小节内容简述每个步骤对应的内容。



尽管我们是以线性和渐进的方式来介绍这 5 个步骤的，但在实践中，颠倒顺序以及同时执行多个步骤都很常见。数据科学是迭代和非线性的过程。例如，数据建模完毕，对结果进行观察后，你可能会决定返回到清洗步骤来调整数据集的特征。

1.2.1　数据获取

没有数据就难以进行数据科学工作。因此第一个步骤就是数据获取。除非你很幸运，已经拥有了相关数据，否则就需要做下列工作。

- 从其他地方（如网页或服务器）下载数据
- 从数据库或 API（如 MySQL 或 Twitter）中查询数据
- 从其他文件（如 HTML 文件或电子表格）中提取数据
- 自己生成数据（例如读取传感器或进行调查）

第 3 章我们讨论使用命令行获取数据的几种方法。获取数据的格式通常是纯文本、CSV、JSON 或 HTML/XML。下一步是数据清洗。

1.2.2　数据清洗

在所获取的数据中，缺失值、不一致、错误、怪异字符或冗余列屡见不鲜。这时，必须清洗数据才能对数据进行后续处理。常见的清洗操作包括：

- 行过滤
- 列抽取
- 值替换
- 单词提取

- 缺失值处理
- 数据格式转换

尽管我们数据科学家热衷于创建激动人心的数据可视化模型和具有洞察力的其他模型（步骤 3 和步骤 4），但通常首先要在获取和清洗所需数据（步骤 1 和步骤 2）上付出大量努力。在“Data Jujitsu”（http://radar.oreilly.com/2012/07/data-jujitsu.html）一文中，DJ Patil 就声称“任何数据项目中 80% 的工作都是数据清洗”（2012）。第 5 章我们将演示怎样用命令行帮助完成这类数据清洗操作。

1.2.3　数据探索

一旦完成数据清洗，就可以准备探索数据了。从这里开始，你会感觉很有趣，因为我们开始真正深入到数据之中。第 7 章将展示如何用命令行进行以下工作：

- 查看数据
- 从数据中推导统计量
- 创建有趣的可视化

第 7 章介绍的命令行工具包括 `csvstat`（Groskopf，2014）、`feedgnuplot`（Kogan，2014）和 `Rio`（Janssens，2014）。

1.2.4　数据建模

如果想解释数据或预测将要发生的事情，你就应该为数据建立一个统计模型。建立模型的技术包括聚类、分类、回归以及降维。命令行不适用于从头开始实现一个新模型，但以命令行为基础建立模型则非常有用。第 9 章我们将介绍若干命令行工具，这些工具有的在本地建立模型，有的使用 API 来执行云端的计算。

1.2.5　数据解释

OSEMN 模型中最后的步骤是数据解释，这可能也是最重要的步骤。这个步骤中包括：

- 从数据中得出结论
- 评估结果的含义
- 告知你的结果

坦白地说，这一阶段中很少用到计算机，实际上也不会运行命令行。一旦到了这一步，工作就该你本人亲自做了。这是 OSEMN 模型中唯一没有单独一章来解释的步骤。我们推荐你参考 Max Shron 的 *Thinking with Data*（由 O’Reilly 于 2014 年出版，http://shop.oreilly.com/product/0636920029182.do）一书。

1.3　插入的几章

在 OSEMN 步骤对应的几章中间，穿插了其他 3 章。每章讨论一个关于数据科学的普遍问题，以及如何用命令行来解决这些问题。这些内容能够适用于数据科学过程中的任何步骤。

第 4 章我们讨论如何创建可重用的命令行工具。这些私人工具可能来自你在命令行上输入的长串命令，也可以源自你已经用 Python 或者 R 编写好的代码。能够创建自己的工具会使你更加高效和多产。

命令行是做数据科学工作的交互式环境，因此跟踪工作流程颇具挑战。在第 6 章，我们将演示一个名为 Drake（Factual，2014）的命令行工具，它使你能够用任务和任务之间依赖的形式来定义数据科学工作流程。这个工具能够增强你自己的工作流程的可再现性，对你的同事和同行也一样有用。

在第 8 章，我们解释怎样通过并行运行来加速命令行和工具。使用名为 GNU Parallel（Tange，2014）的命令行工具，可以将命令行工具应用于海量的数据集，并在多核和远程机器上运行。

1.4　什么是命令行

在讨论为什么应该在数据科学中使用命令行之前，让我们先看一下命令行的庐山真面目（或许你已经熟悉）。图 1-1 和图 1-2 分别展示了默认情况下命令行在 Mac OS X 和 Ubuntu 上的屏幕截图。Ubuntu 是 GUN/Linux 的一个特殊的发行版，本书始终默认使用该系统。

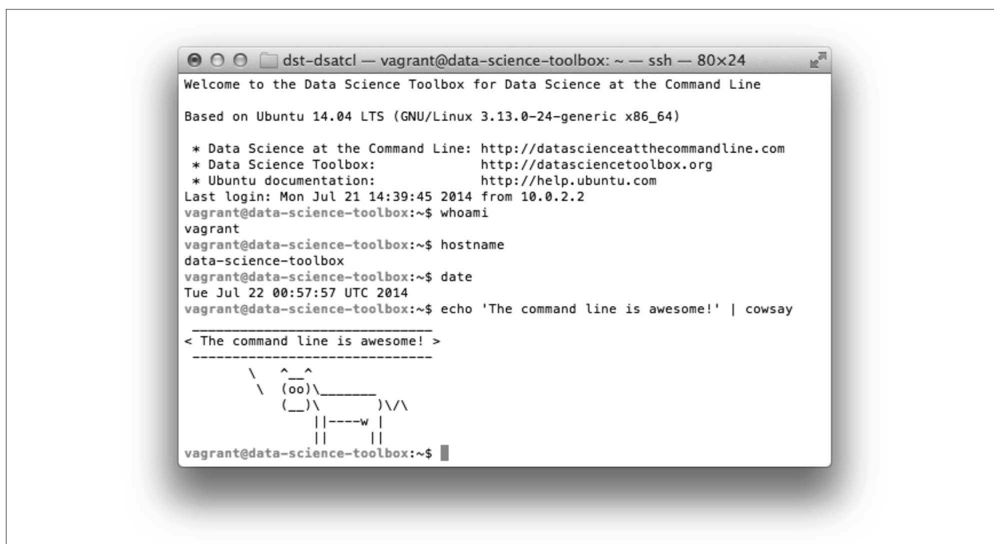


图 1-1：Mac OS X 上的命令行

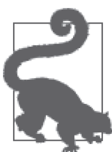
```
vagrant@data-science-toolbox: ~
Welcome to the Data Science Toolbox for Data Science at the Command Line

Based on Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Data Science at the Command Line: http://datascienceatthecommandline.com
 * Data Science Toolbox:             http://datasciencetoolbox.org
 * Ubuntu documentation:             http://help.ubuntu.com
Last login: Mon Jul 21 01:11:59 2014 from 10.0.2.2
vagrant@data-science-toolbox:~$ whoami
vagrant
vagrant@data-science-toolbox:~$ hostname
data-science-toolbox
vagrant@data-science-toolbox:~$ date
Mon Jul 21 01:29:04 UTC 2014
vagrant@data-science-toolbox:~$ echo 'The command line is awesome!' | cowsay
 ______________________________
< The command line is awesome! >
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
vagrant@data-science-toolbox:~$
```

图 1-2：Ubuntu 上的命令行

这两个截图中的窗体称为终端，是帮助你与 shell 交互的程序。我们输入的命令都是由 shell 执行的（Ubuntu 和 Mac OS X 的默认 shell 都是 Bash）。



我们没有给出微软 Windows 系统的命令行（也就是命令提示符或 PowerShell），这是因为它与本书中给出的命令行有着根本上的不同，并且也不兼容。不过，好消息是可以在微软 Windows 上安装数据科学工具箱，这样你仍然能够跟随本书学习。我们将在第 2 章介绍怎样安装数据科学工具箱。

需要输入命令，这是与图形用户界面截然不同的人机交互方式。如果你习惯于使用微软 Excel 来处理数据，那么刚开始命令行可能显得很烦人。请不要担心，我保证你很快就会习惯的。

本书中，我们输入的命令及其输出都以文本的形式显示。例如，以上两个屏幕截图中，终端上的内容（欢迎信息之后）将会是：

```
$ whoami
vagrant
$ hostname
data-science-toolbox
$ date
Tue Jul 22 02:52:09 UTC 2014
$ echo 'The command line is awesome!' | cowsay
```

```
 _____________________________
< The command line is awesome! >
 -----------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

你可能还会注意到每个命令前面都有一个美元符号（$），这叫作提示符。以上两个屏幕截图中的提示符包含了更多的信息，包括用户名（vagrant）、主机名（data-science-toolbox）和当前的工作目录（~）。在例子中使用美元符号是惯例，因为这个提示符既能够在会话期间改变（当你进入另外的目录时），又能够由用户定制（例如，它还能够显示时间或你当前所在的 git（Torvalds & Hamano，2014）分支，还可以与命令本身无关。

下一章我们将对基本的命令行概念做出更多解释。现在，首先来解释一下为什么你应该学习使用命令行来做数据科学工作。

1.5 为什么用命令行做数据科学工作

命令行有许多显著的优点，能使你真正成为更加高效和多产的数据科学家。粗略地总结一下，这些优点包括：灵活、可增强、可扩展、可扩充以及无所不在。下面详细阐述每条优点。

1.5.1 命令行的灵活性

命令行的第一个优点是它能赋予你灵活性。数据科学具有很强的交互性和探索性，你的数据科学工作环境应该支持这样的特性。命令行通过两种方法来达到这个目的。

首先，它提供了称为读取－求值－打印－循环（read-eval-print-loop，REPL）的机制。这意味着输入一个命令，按下回车键，这个命令就立即被执行得出结果。在数据科学中，REPL 经常比编辑－编译－运行－调试循环方便得多，后者通常适用于脚本、大型程序以及 Hadoop 任务。命令立即就会执行，也可以随意停止，还可以迅速修改。这种简短的迭代循环能使你真正享受鼓捣数据的乐趣。

其次，它与文件系统的距离很近。数据是数据科学工作的主要原料，因此很重要的一点是要能很容易地处理包含数据集的文件。命令行为此提供了许多方便的工具。

1.5.2 命令行可增强

不论目前你的数据科学工作流程使用了哪种技术（R、IPython 还是 Hadoop），你都应该明白我们并不建议抛弃这些工作流程。相反，这里讲到的命令行技术扮演了倍增器的角色，能够增强你现有技术的威力。

命令行与其他技术能够无缝集成。一方面，你可以经常在自己的工作环境中采用命令行。例如，可以在 Python 和 R 中运行命令行工具并捕获其结果。另一方面，也可以将自己的代码（例如以前编写的 Python 或 R 函数）转变为命令行工具。我们将在第 4 章详细讨论这一问题。此外，命令行易于和各种数据库及文件类型（如微软 Excel）协同工作。

当然，任何技术都有优点和不足（包括命令行），因此最好是通晓多种技术，根据具体任务使用最适合的技术。有时应使用 R，有时是命令行，有时甚至是笔和纸。看完本书，你将深刻理解什么时候可以使用命令行，以及什么时候继续使用你喜欢的编程语言或统计计算环境更好。

1.5.3 命令行可扩展

使用命令行与使用图形用户界面（GUI）是截然不同的。命令行是通过输入命令来使用的，GUI 则是通过鼠标指向和点击来使用的。

在命令行上人工输入的所有任务，都可以通过脚本和工具自动化。这样，在发生错误、数据集改变或你的同事需要执行相同分析时，可以很容易地重新执行命令。此外，命令可以在远程服务器上以指定的时间间隔执行，还可以在大量数据上并行执行（第 8 章进一步阐述）。

由于命令行是可以自动化的，因此它是可扩展和可重复的。相反，要将鼠标指向和点击自动化则没那么简单，因此对于需要可扩展性和可重复性的数据科学而言，GUI 环境就没那么合适了。

1.5.4 命令行可扩充

命令行是在 40 多年前发明的。它的核心功能绝大部分未曾改变，但是作为命令行的主力的命令行工具，几乎每天都有新的出现。

命令行本身是不依赖于语言的。这使得命令行工具的编写语言五花八门。开源社区提供了许多可以用于数据科学的命令行工具，它们都免费并且质量很高。

这些工具能够一起工作，这使得命令行非常灵活。你也可以创建属于自己的工具，从而使你能够扩展命令行的有效功能。

1.5.5 命令行无处不在

包括 Ubuntu 和 Mac OS X 在内的所有类 Unix 操作系统都自带有命令行，因此在许多计算机上都能找到它。根据 Top 500 超级计算机网站上的文章（http://top500.org/blog/lists/2013/11/press-release），Top 500 超级计算机中 95% 运行的是 GNU/Linux。因此，如果你要接触这些超级计算机（或者你想逃离侏罗纪公园，却发现门锁坏了），最好熟悉命令行。

GNU/Linux 不仅仅在超级计算机上运行，还运行在服务器、笔记本电脑以及嵌入式系统中。现在，许多企业提供云计算服务，你可以很容易地联机启动新的机器。如果你曾经登录进入这样的机器（或者更一般的服务器），那么很可能会接触到命令行。

在讨论了命令行的无所不在之后，需要重点强调的是，命令行绝不是过眼烟云似的炒作。这项技术已经有 40 多年的历史，我个人相信它还可以再持续 40 年。因此，学习如何使用命令行（用于数据科学）是一项物有所值的投资。

1.6 一个现实用例

前面我们给出了数据科学的定义，解释了为什么命令行是进行数据科学工作的优异环境。现在，该通过一个现实用例来展示命令行的能力和灵活性了。我们将快刀斩乱麻，如果有不理解的地方，请先不必担心。

我个人似乎从来不记得纽约的时装周什么时候开幕。我只知道每年举办两次，但每次举办都让我出乎意料！本节我们将通过查阅《纽约时报》好用的 Web API，来计算时装周的举办时间。首先要在开发者网站（http://developer.nytimes.com）取得自己的 API 密钥，这样就能进行搜索文章、获取畅销列表以及查看事件清单这样的操作。

我们将要查询的特定 API 端点是文章搜索。我们期望通过《纽约时报》对纽约时装周的报道数量的高峰，判断它是否开始。API 的结果是分页的，这意味着必须以不同的页码多次执行相同的查询（这类似于在搜索引擎上点击“下一页”按钮）。这时 GNU Parallel（Tange，2014）就可以大显身手，因为它可以充当 `for` 循环。所需要的全部命令如下所示（不要担心 `parallel` 的那一长串命令行参数，我们将会在第 8 章详细讨论）：

```
$ cd ~/book/ch01/data
$ parallel -j1 --progress --delay 0.1 --results results "curl -sL "\
> "'http://api.nytimes.com/svc/search/v2/articlesearch.json?q=New+York+'"\
> "'Fashion+Week&begin_date={1}0101&end_date={1}1231&page={2}&api-key='"\
> "'<your-api-key>'" ::: {2009..2013} ::: {0..99} > /dev/null

Computers / CPU cores / Max jobs to run
1:local / 4 / 1

Computer:jobs running/jobs completed/%of started jobs/Average seconds to
complete
local:1/9/100%/0.4s
```

实际上，要对 2009~2014 年执行相同的查询。对每个查询，API 最多返回 100 页（从 0 开始计数），因此使用大括号扩展生成 100 个数字。查询中的“page”参数使用这些数字。现在我们要查找 2013 年包含搜索项“New+York+Fashion+Week”的文章。由于 API 有一定的限制，需要确保每次只有一个请求，请求之间有 1 秒的延迟。这里，要确保将 `<your-api-key>` 替换为自己文章搜索端点的 API 密钥。

每个请求返回 10 篇文章，因此总共有 1000 篇文章。文章通过页面访问量排序，据此可以对报道做一个很好的估计。请求结果是 JSON 格式，存储在 result 目录中。通过命令行工具 `tree`（Baker，2014）可以查看子目录结构：

```
$ tree results | head
results
└── 1
    ├── 2009
    │   └── 2
    │       ├── 0
    │       │   ├── stderr
    │       │   └── stdout
    │       ├── 1
    │       │   ├── stderr
    │       │   └── stdout
```

现在使用 `cat`（Granlund & Stallman，2012）、`jq`（Dolan，2014）和 `json2csv`（Czebotar，2014）来合并和处理结果：

```
$ cat results/1/*/2/*/stdout |                                         ❶
> jq -c '.response.docs[] | {date: .pub_date, type: .document_type, '\  ❷
> 'title: .headline.main }' | json2csv -p -k date,type,title > fashion.csv ❸
```

把这条命令分解开来：

❶ 将每 500 个 `parallel` 任务（或 API 请求）的结果合并起来。
❷ 使用 `jq` 抽取出版日期、文档类型以及每篇文章的标题。
❸ 使用 `json2csv` 将 JSON 数据转换为 CSV 格式，将其存为 fashion.csv。

通过使用 `wc -l`（Rubin & MacKenzie，2012），我们发现数据集包含 4855 篇文章（不是 5000 篇，因为之前我们很可能提取了从 2009 年开始的所有东西）：

```
$ wc -l fashion.csv
4856 fashion.csv
```

现在，让我们检查前 10 篇文章，来验证是否已经成功获取了数据。需要注意的是，我们在 date 列上使用 `cols`（Janssens，2014）和 `cut`（Ihnat、MacKenzie & Meyering，2012）来删掉表中的时间和时区信息：

```
$ < fashion.csv cols -c date cut -dT -f1 | head | csvlook
|-------------+------------+-----------------------------------------|
| date        | type       | title                                   |
|-------------+------------+-----------------------------------------|
| 2009-02-15  | multimedia | Michael Kors                            |
| 2009-02-20  | multimedia | Recap: Fall Fashion Week, New York      |
| 2009-09-17  | multimedia | UrbanEye: Backstage at Marc Jacobs      |
| 2009-02-16  | multimedia | Bill Cunningham on N.Y. Fashion Week    |
| 2009-02-12  | multimedia | Alexander Wang                          |
```

```
| 2009-09-17  | multimedia | Fashion Week Spring 2010                  |
| 2009-09-11  | multimedia | Of Color | Diversity Beyond the Runway  |
| 2009-09-14  | multimedia | A Designer Reinvents Himself              |
| 2009-09-12  | multimedia | On the Street | Catwalk                   |
|-------------+------------+-------------------------------------------|
```

看起来已经大功告成！为了尽可能加深理解，最好将数据可视化。图 1-3 中是用 R（R Foundation for Statistical Computer，2014）、`Rio`（Janssens，2014）和 `ggplot2`（Wickham，2009）生成的线形图。

```
$ < fashion.csv Rio -ge 'g + geom_freqpoly(aes(as.Date(date), color=type), '\
> 'binwidth=7) + scale_x_date() + labs(x="date", title="Coverage of New York'\
> 'Fashion Week in New York Times")' | display
```

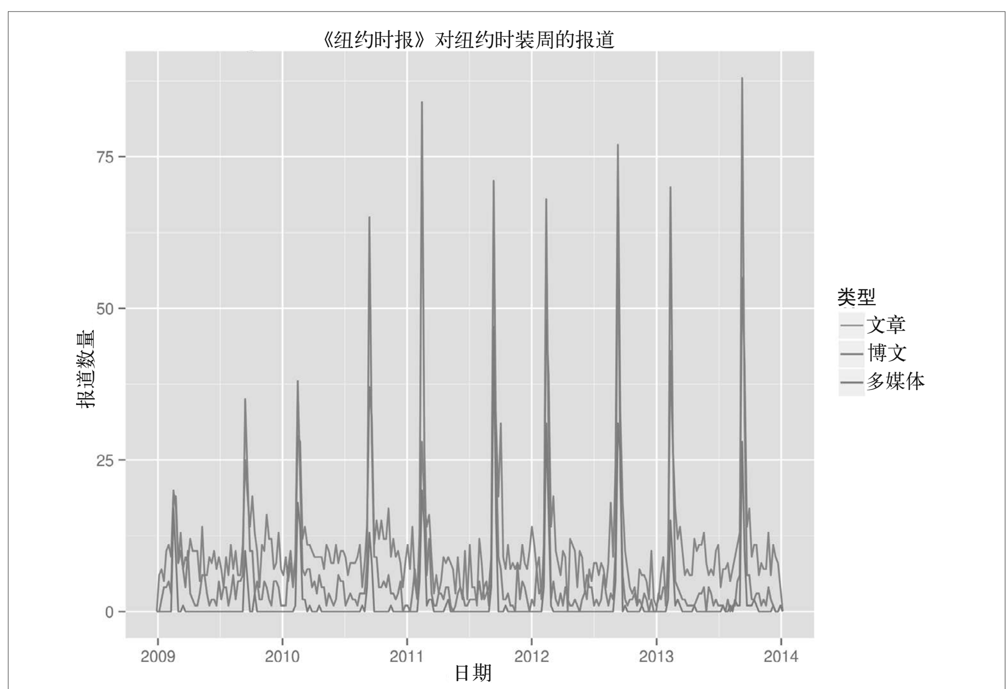


图 1-3：《纽约时报》关于纽约时装周的报道

通过查看线形图，可以推断纽约时装周每年举办两次。现在我们知道具体时间了：一次在 2 月，另一次在 9 月。假设今年也将在同样的时间举办，这样我们就可以做好准备了。不管怎样，希望这个例子能告诉你，《纽约时报》API 是一个有趣的数据源。更重要的是，希望你相信，命令行是进行数据科学工作的非常棒的方法。

本节我们简单浏览了一些重要概念和激动人心的命令行工具。如果你现在还没有体会到某些东西的价值，请不要担心。第 2 章将讨论大部分概念，接下来的几章要对本节中使用的所有命令行工具做更详细的介绍。

1.7 延伸阅读

- Mason, H., & Wiggins, C. H. (2010). A Taxonomy of Data Science. Retrieved May 10, 2014, from http://www.dataists.com/2010/09/a-taxonomy-of-data-science.
- Patil, D (2012). Data Jujitsu. O'Reilly Media.
- O'Neil, C., & Schutt, R. (2013). *Doing Data Science*. O'Reilly Media.
- Shron, M. (2014). *Thinking with Data*. O'Reilly Media.

第 2 章 入门指南

本章将确保你具备使用命令行进行数据科学工作的全部先决条件。这些条件分为两个部分：(1) 建立一个合适的环境，包含本书中所使用的所有命令行工具。(2) 理解使用命令行时涉及的必要概念。

首先介绍怎样安装数据科学工具箱，这是一个基于 GNU/Linux 的虚拟环境，包含了所有必要的命令行工具。接下来举例解释必要的命令行概念。

读完本章，你将具备继续进行数据科学工作的第 1 个步骤（即数据获取）所需要的全部东西。

2.1 概述

本章你将学到：

- 怎样设置数据科学工具箱
- 用命令行进行数据科学工作所需掌握的概念和工具

2.2 设置数据科学工具箱

书中使用了许多不同的命令行工具。我们使用的 GNU/Linux 发行版 Ubuntu 自带了一大堆预安装的命令行工具，并且还提供了许多包含其他相关命令行工具的程序包。自行安装这些程序包并不是很困难的事。不过，我们还用到一些无法以程序包形式提供的命令行工具，安装这些工具需要更多的人工操作，也更复杂一些。为了获得必要的命令行工具而不

经历复杂的安装过程，我们鼓励你安装数据科学工具箱。



如果你选择在本地而不是虚拟机中运行命令行工具，那就可以逐个安装命令行工具，不过请注意这会非常耗时。附录 A 列出了书中使用的全部命令行工具。安装说明只是针对 Ubuntu 的，如果要在其他操作系统上安装本地的命令行工具，请到本书的网站（http://datascienceatthecommandline.com/）查找最新信息。书中使用的脚本和数据集可以从本书的 GitHub 资源库（https://github.com/jeroenjanssens/data-science-at-the-command-line）中复制得到。

数据科学工具箱是一个虚拟环境，它使你能够在几分钟内开始数据科学工作。默认版本自带有常用的数据科学软件，包括 Python 科学计算栈以及带有其最流行程序包的 R 语言。额外的软件和数据包很容易安装。这些包可以是某本书、某门课程或某个组织所特有的。在数据科学工具箱网站（http://datasciencetoolbox.org/）上可以了解到更多信息。

设置数据科学工具箱有两种方式：(1) 使用 VirtualBox 和 Vagrant 在本地安装；(2) 使用亚马逊 Web 服务在云上启动。两种方式生成的环境是完全相同的。本章介绍怎样在本地为基于命令行的数据科学设置数据科学工具箱。如果你希望在云上运行它，或者遇到问题，请参考本书的网站。

最简单的方式就是将数据科学工具箱安装在本地机器上。由于数据科学工具箱的本地版本是在 VirtualBox 和 Vagrant 上运行，因此可以将其安装到 Linux、Mac OS X 和微软 Windows 上。

2.2.1　步骤1：下载和安装VirtualBox

进入 VirtualBox（Oracle，2014）的下载页面（https://www.virtualbox.org/wiki/Downloads），下载适合你的操作系统的二进制安装文件。打开安装文件，按照安装说明来操作。

2.2.2　步骤2：下载和安装Vagrant

与步骤 1 类似，进入 Vagrant（HashiCorp，2014）的下载页面（http://www.vagrantup.com/downloads.html），下载相应的二进制安装文件。打开安装文件，按照安装说明进行操作。如果以前安装过 Vagrant，要确保其版本号不低于 1.5。

2.2.3　步骤3：下载并启动数据科学工具箱

打开终端（微软 Windows 中称为命令提示符或 PowerShell）。创建一个目录，例如 MyDataScienceToolbox，输入下列命令进入该目录：

```
$ mkdir MyDataScienceToolbox
$ cd MyDataScienceToolbox
```

运行下列命令初始化数据科学工具箱：

```
$ vagrant init data-science-toolbox/data-science-at-the-command-line
```

这就创建了一个名为 Vagrantfile 的文件。这是一个配置文件，用来告诉 Vagrant 怎样启动虚拟机。它包含许多被注释掉的代码。最精简的版本如示例 2-1 所示。

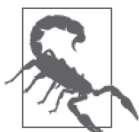
示例 2-1　Vagrant 的最精简配置

```
Vagrant.configure(2) do |config|
  config.vm.box = "data-science-toolbox/data-science-at-the-command-line"
end
```

运行下列命令，就可以下载并启动数据科学工具箱：

```
$ vagrant up
```

如果一切顺利，现在你就拥有了一个运行于本地机器上的数据科学工具箱了。



如果你看到了以下消息重复显示：`default: Warning: Connection time out. Retrying...`，可能是虚拟机在等待输入。这可能是因为虚拟机没有正确关闭。为了查出问题所在，在 Vagrantfile 末尾的 `end` 语句前加入下列代码（也可参考示例 2-2）：

```
config.vm.provider "virtualbox" do |vb|
  vb.gui = true
end
```

这会使 VirtualBox 显示出界面。一旦虚拟机完成启动，并且你已找出问题所在，就可以将这些代码从 Vagrantfile 中删除。登录所需要的用户名和密码都是 `vagrant`。如果问题仍然无法解决，建议你查一下本书的网站，里面包含有常见问题的最新清单。

示例 2-2 给出了一个稍显复杂的 Vagrantfile。你可以在 http://docs.vagrantup.com 上查看更多的配置选项。

示例 2-2　配置 Vagrant

```
Vagrant.require_version ">= 1.5.0"                                        ❶
Vagrant.configure(2) do |config|
  config.vm.box = "data-science-toolbox/data-science-at-the-command-line"
  config.vm.network "forwarded_port", guest: 8000, host: 8000             ❷
  config.vm.provider "virtualbox" do |vb|
    vb.gui = true                                                          ❸
    vb.memory = 2048                                                       ❹
    vb.cpus = 2                                                            ❺
  end
end
```

❶ Vagrant 的版本号不低于 1.5.0。
❷ 转发端口号为 8000。这在你想要查看所创建的图形时很有用，对此第 7 章会有所涉及。
❸ 启动一个图形用户界面。
❹ 使用 2 GB 的内存。
❺ 使用 2 个 CPU。

2.2.4　步骤4：登录（Linux和Mac OS X）

如果你运行 Linux、Mas OS X 或其他类 Unix 操作系统，就可以通过在终端中运行下列命令来登录数据科学工具箱：

```
$ vagrant ssh
```

几秒钟之后，你就可以看到下列欢迎消息：

```
Welcome to the Data Science Toolbox for Data Science at the Command Line

Based on Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Data Science at the Command Line: http://datascienceatthecommandline.com
 * Data Science Toolbox:             http://datasciencetoolbox.org
 * Ubuntu documentation:             http://help.ubuntu.com
Last login: Tue Jul 22 19:33:16 2014 from 10.0.2.2
```

2.2.5　步骤4：登录（微软Windows）

如果你运行微软 Windows，就需要用图形用户界面（关于如何设置请参考前面的步骤 2），或者使用第三方应用程序，来运行 Vagrant，这样才能登录数据科学工具箱。对于后面这种情况，我们推荐你使用 PuTTY。进入 PuTTY 的下载页面（http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html），下载 putty.exe。运行 PuTTY，输入下列数值：

- 主机名（或 IP 地址）：`127.0.0.1`
- 端口号：`2222`
- 连接类型：`SSH`

如果需要，可以点击保存按钮将这些值存为一个会话，这样下次就不需要再次输入这些值了。点击打开按钮，在用户名和密码处都输入 `vagrant`。

2.2.6　步骤5：关闭或重启

可以在与运行 `vagrant up` 命令相同的目录下运行下列命令来关闭数据科学工具箱：

```
$ vagrant halt
```

如果希望关闭数据科学工具箱，然后再次启动，可以输入：

```
$ vagrant destroy
```

然后，按照前面步骤 3 的方法再次设置数据科学工具箱。

2.3 必要的概念和工具

第 1 章简要介绍了什么是命令行。既然你拥有了自己的数据科学工具箱，我们就可以真正开始了。本节我们讨论为了得心应手地用命令行进行数据科学工作，所需要了解的几个概念和工具。如果到目前为止你主要使用图形用户界面，这可能是一个巨大转变。但是也不必担心，我们将从基础讲起，一点点地逐步进阶到高级主题。



本节不是 GNU/Linux 的完整教程。我们只解释与基于命令行的数据科学工作相关的概念和工具。数据科学工具箱的一个优势在于大量东西都是现成的。如果你希望更多地了解 GNU/Linux，参见本章末尾的“延伸阅读”。

2.3.1 环境

现在你刚刚登录进入一个全新的环境。在一切工作开始之前，详细了解一下环境是很有必要的。这个环境可以由四个层次来粗略定义，我们自上而下简要讨论。

- 命令行工具

  首先是你使用的命令行工具。我们通过输入对应的命令来使用它们。命令行工具有不同的类型，这将在下一章中讨论。举例来说，这些工具包括 `ls`（Stallman & MacKenzie，2012）、`cat`（Granlund & Stallman，2012），以及 `jq`（Dolan，2014）。

- 终端

  位于第二层的终端是我们输入命令的应用程序。如果看到下列文本：

  ```
  $ seq 3
  1
  2
  3
  ```

  就意味着你向终端中输入了 `seq 3`，然后按下了回车键。[命令行工具 `seq`（Drepper，2012）生成一个数字序列。] 美元符号不是你输入的，它只是作为提示符让你知道可以输入这个命令。`seq 3` 下面的文本就是命令的输出。第 1 章中，我们用两幅截图展示了在 Mac OS X 和 Ubuntu 上运行各种命令及其输出时，终端是什么样的。

- shell

  第三层是 shell。一旦输入了命令并按下回车键，终端就将命令送给 shell。shell 是对命令进行解释的程序。数据科学工具箱使用 Bash 作为 shell，不过还有许多其他 shell 可以使用。一旦对命令行略有心得，你可能会对称为 Z shell 的 shell 感兴趣。它提供了许多额外的特性，能够提高你使用命令行的工作效率。

- 操作系统

  第四层是操作系统，本书指的就是 GNU/Linux。Linux 是内核的名字，它是操作系统的核心。内核与 CPU、磁盘以及其他硬件直接交互。内核也运行我们的命令行工具。GNU 是 GNU's Not Unix 的缩写，指的是一系列基础工具。数据科学工具箱基于的是称为 Ubuntu 的 Linux 特殊发行版。

2.3.2　运行命令行工具

既然你对环境有了一定的理解，现在就可以来尝试某些命令了。在终端中输入下列命令（除了美元符号），然后按下回车键：

```
$ pwd
/home/vagrant
```

这个命令很简单，只包含单个命令行工具。命令行工具 `pwd`（Meyering，2012）用于打印出当前所在目录的名字。默认情况下，这就是你登录进入的主目录。可以用 `ls`（Stallman & MacKenzie，2012）命令来查看目录的内容：

```
$ ls
book
```

使用 Bash 内置的命令行工具 `cd`，可以进入其他目录：

```
$ cd book/ch02/
$ cd data
$ pwd
/home/vagrant/book/ch02/data
$ cd ..
$ pwd
/home/vagrant/book/ch02/
```

`cd` 后面的部分指定想要进入的目录。紧跟在命令后面的值称为命令行参数或选项。两个点指的是父目录。让我们再尝试另一条命令：

```
$ head -n 3 data/movies.txt
Matrix
Star Wars
Home Alone
```

这里向 head 命令（MacKenzie & Meyering，2012）传递了三个命令行参数。第一个参数是选项，第二个是该选项的值，第三个是文件名。这条命令输出文件 ~book/ch02/data/movies.txt 的前三行。

有时使用的命令和管道太长，无法在页面中显示完整。这时，你会看到类似下面的情况：

```
$ echo 'Hello'\
> ' world' |
> wc
```

大于符号（>）是继续提示符，意味着这一行是前一行的继续。长命令可以用反斜杠（\）或管道符号（|）分割开。一定要首先匹配所有标有 " 和 ' 的引用。下列命令就与前面的命令完全相同。

```
$ echo 'Hello world' | wc
```

2.3.3 五类命令行工具

前面大量使用“命令行工具”这个术语，但还没有解释它的实际含义。实际上，我们用它统称从命令行执行的任何东西。具体而言，命令行工具都可以归属为下列五种类型：

- 二进制可执行文件
- shell 内置命令
- 解释型脚本
- shell 函数
- 别名

了解不同类型间的差异是有好处的。数据科学工具箱中预安装的命令行工具大多属于前两类（二进制可执行文件和 shell 内置命令）。其他三类（解释型脚本、shell 函数和别名）可以进一步增强我们自己的数据科学工具箱 [1]，帮助我们成为更加高效和多产的数据科学家。

- 二进制可执行文件

  二进制可执行文件是传统意义上的程序。它是由源代码编译为机器代码而成的。这意味着在文本编辑器中打开这类文件时，你无法看到其源代码。

- shell内置命令

  shell 内置命令是由 shell 提供的命令行工具，本书中是由 Bash 提供的。cd 和 help 就是其中的例子。它们是无法更改的。不同 shell 的内置命令可能有差异。与二进制可执行文件类似，它们都难以被直接查看或修改。

注 1：这里指的不是字面上的数据科学工具箱，而是用比喻的方式说明要有自己的工具集。

- 解释型脚本

  解释型脚本是由二进制可执行文件所执行的文本文件，例如 Python、R 和 Bash 脚本。解释型脚本的一个很大的优点是你可以读懂和修改它。示例 2-3 展示了名为 ~/book/ch02/fac.py 的脚本。它是由 Python 解释的，这并不是因为它的文件扩展名是 .py，而是由于脚本的第一行指定了执行它的二进制可执行文件。

  示例 2-3　计算整数阶乘的 Python 脚本（~/book/ch02/fac.py）

  ```
  #!/usr/bin/env python

  def factorial(x):
      result = 1
      for i in xrange(2, x + 1):
          result *= i
      return result

  if __name__ == "__main__":
      import sys
      x = int(sys.argv[1])
      print factorial(x)
  ```

  该脚本计算一个整数的阶乘，这个整数是传递的命令行参数。可以通过下列方式在命令行中调用该脚本：

  ```
  $ book/ch02/fac.py 5
  120
  ```

  第 4 章将详尽讨论怎样使用解释型脚本来创建可重用的命令行工具。

- shell函数

  shell 函数是由 shell 自己执行的函数，在本书中就是由 Bash 执行的。它们提供了与 Bash 脚本相似的功能，但通常（尽管不是必然的）比脚本小一些，也更趋于个人化。下列命令定义了一个称为 `fac` 的函数，与前文的 Python 解释型脚本类似，用于计算作为参数传递给函数的某个整数的阶乘。具体而言，该函数使用 `seq` 来生成一串数字，用 `paste`（Ihnat & MacKenzie，2012）将这些数字放到一行中并用 * 分隔开，然后将这个等式传递给 `bc`（Nelson，2006），由它求值并输出结果：

  ```
  $ fac() { (echo 1; seq $1) | paste -s -d\* | bc; }
  $ fac 5
  120
  ```

  文件 ~/.bashrc 是 Bash 的配置文件，它是定义 shell 函数的好地方，这样 shell 函数就随时可用。

- 别名

  别名与宏类似。如果你发现自己经常要用相同（或一部分相同）的参数执行一个特定的

命令，那就可以为此定义一个别名。在你总是拼错特定命令时，别名也很有用（在该命令的 GitHub 配置文件 https://github.com/chrishwiggins/mise/blob/master/sh/aliases-public.sh 中，你可以看到一个很长的清单，列出了许多有用的别名）。下列命令定义了两个别名：

```
$ alias l='ls -1 --group-directories-first'
$ alias moer=more
```

现在，如果你在命令行中进行如下输入，shell 将把它找到的每个别名替换为对应的值：

```
$ cd ~
$ l
book
```

别名要比 shell 函数简单一些，因为它们没有参数。函数 `fac` 无法用别名来定义，因为它有参数。别名还能使你大量减少敲击键盘的次数。类似于 shell 函数，别名经常是在主目录中的 .bashrc 或 .bash_aliases 配置文件中定义的。要看当前定义的所有别名，不带参数地运行 `alias` 即可。试一下，看看会发生什么？

本书中，当提到创建新的命令行工具时，我们主要关注后三种类型：解释型脚本、shell 函数和别名。这是因为它们易于修改。命令行工具的作用是使你用命令行工作起来更轻松，成为更加多产和高效的数据科学家。你可以使用 `type`（它本身就是一个 shell 内置命令）来查看命令行工具的类型：

```
$ type -a pwd
pwd is a shell builtin
pwd is /bin/pwd
$ type -a cd
cd is a shell builtin
$ type -a fac
fac is a function
fac ()
{
    ( echo 1;
    seq $1 ) | paste -s -d\* | bc
}
$ type -a l
l is aliased to `ls -1 --group-directories-first'
```

如你所见，`type` 对 `pwd` 返回了两个命令行工具。第一个描述的是输入 `pwd` 时所使用的命令行工具。下一节，我们来看看怎样将命令行工具组合起来。

2.3.4 命令行工具的组合

多数命令行工具遵循 Unix 哲学，即只做一件事情，并且做得十分出色。例如，命令行工具 `grep`（Meyering，2012）对行进行过滤，`wc`（Rubin & MacKenzie，2012）计算行数，`sort`（Haertel & Eggert，2012）对行进行排序。命令行的力量在于它能将这些小而强大的

命令行工具组合起来。最常见的方式是通过管道进行组合。一个工具的输出传递给下一个工具，这样几乎没有限制地不断传递。

例如，考虑命令行工具 seq，该工具生成一个数字序列。让我们生成 5 个数字的序列：

```
$ seq 5
1
2
3
4
5
```

默认情况下，命令行工具的输出是传递给终端的，由终端显示在显示器屏幕上。我们可以将 seq 的输出通过管道传递给下一个称为 grep 的工具，它用来对行进行过滤。假设我们只想看到包含 3 的数字，可以用如下方式将 seq 和 grep 组合起来：

```
$ seq 30 | grep 3
3
13
23
30
```

如果想要知道 1 和 100 之间有多少数字包含 3，可以使用 wc，它特别擅长计数：

```
$ seq 100 | grep 3 | wc -l
19
```

选项 -l 指定了 wc 只输出行的数量。默认情况下 wc 还返回字符和单词的数量。

你可能开始认识到组合命令行工具是一个非常强大的概念。后面，我们将向你介绍更多的工具，以及将它们组合起来能提供什么功能。

2.3.5 输入和输出重定向

我们曾经提到过，默认情况下，管道中最后一个命令行工具是输出到终端的。你也可以将这个输出保存到文件中。这称为输出重定向，通过如下方式进行：

```
$ cd ~/book/ch02
$ seq 10 > data/ten-numbers
```

这里，我们将 seq 工具的输出保存到目录 ~/book/ch02/data 中名为 ten-numbers 的文件中。如果这个文件不存在，就会新创建出来；如果文件已经存在，它的内容将会被覆盖。也可以用 >> 将输出附加到文件末尾，意思是将输出附加到原来内容的后面：

```
$ echo -n "Hello" > hello-world
$ echo " World" >> hello-world
```

工具 echo 只是输出指定的值。选项 -n 指定 echo 不要输出尾部新行。

如果你需要存储中间结果（例如用来在后面的阶段继续进行分析），将输出保存到文件中会很有用。要使用文件 hello-world 的内容，可以用 cat（Granlund & Stallman，2012）来读取文件并显示出来：

```
$ cat hello-world | wc -w
2
```

（注意，选项 -w 表明 wc 只计算单词的数量。）也可以用下列记号获得相同的结果：

```
$ < hello-world wc -w
2
```

这样，不需要运行一个额外的进程就可以直接将文件传递给 wc 的标准输入。如果命令行工具还允许将文件指定为命令行参数（事实上许多工具都是这样的），还可以这样运行 wc：

```
$ wc -w hello-world
2 hello-world
```

2.3.6 处理文件

作为数据科学家，我们的工作对象是大量数据，它们通常存储在文件之中。因此，知道怎样用命令行来处理文件（以及文件所在的目录）是很重要的。用图形用户界面能做的任何操作，都可以用命令行工具来完成（而且能做到更多）。本节我们将介绍最重要的几个操作，即对文件和目录进行创建、移动、复制、重命名和删除。

你已经知道了怎样用 > 或 >> 来对输出进行重定向，从而创建新文件。如果需要将文件移动到其他目录，可以使用 mv（Parker，MacKenzie & Meyering，2012）：

```
$ mv hello-world data
```

也可以用 mv 将文件重命名：

```
$ cd data
$ mv hello-world old-file
```

还可以重命名或移动整个目录。如果不再需要某个文件，可以用 rm（Rubin，MacKenzie，Stallman & Meyering，2012）删除（或移除）它：

```
$ rm old-file
```

如果想要删除整个目录及其包含的全部内容，那就指定选项 -r，意为递归删除：

```
$ rm -r ~/book/ch02/data/old
```

如果想复制文件，使用 `cp`（Granlund，MacKenzie & Meyering，2012）。这在创建备份时很有用：

```
$ cp server.log server.log.bak
```

可以使用 `mkdir`（MacKenzie，2012）创建目录：

```
$ cd data
$ mkdir logs
```

所有这些命令行工具都接受选项 `-v`，意为“详细”（verbose），能使工具输出正在进行的动作。除了 `mkdir` 之外的其他所有工具都接受选项 `-i`，意为“交互”（interactive），能让工具向你请求确认。

刚开始使用命令行工具来管理文件可能会令你望而生畏，这是因为对文件系统缺少图形化的总体印象，会导致你无法提供直接的反馈。

2.3.7 寻求帮助

在命令行的学习中上下求索时，你有时可能需要帮助。有时，即使最老到的 Linux 用户也会需要帮助。记住所有的命令行工具及所有选项显然是不可能的。幸好，命令行提供了几种方法来获取帮助。

获取帮助最重要的命令可能就是 `man`（Eaton & Watson，2014）了，它是“manual”的缩写，包含了绝大多数命令行工具的信息。假设我们忘记了工具 `cat` 的各种选项，可以使用下列命令来访问它的操作说明页面（Man Page）：

```
$ man cat | head -n 20
CAT(1)                          User Commands                          CAT(1)


NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s), or standard input, to standard output.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonempty output lines, overrides -n

       -e     equivalent to -vE
```



有时你会发现我们在命令的末尾使用 head、fold 或 cut。这只是为了确保命令的输出适合页面宽度，你不需要键入它们。例如，head -n 5 是只显示前 5 行，fold 将较长的行变为 80 字符长度，cut -c1-80 则裁剪长度超过 80 字符的行。

不是所有的命令行工具都有操作说明页面。对于像 cd 这样的 shell 内置命令，需要使用 help 命令行工具：

```
$ help cd | head -n 20
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR. The default DIR is the value of the
    HOME shell variable.

    The variable CDPATH defines the search path for the directory containing
    DIR. Alternative directory names in CDPATH are separated by a colon (:).
    A null directory name is the same as the current directory. If DIR begins
    with a slash (/), then CDPATH is not used.

    If the directory is not found, and the shell option 'cdable_vars' is set,
    the word is assumed to be  a variable name.  If that variable has a value,
    its value is used for DIR.

    Options:
        -L      force symbolic links to be followed: resolve symbolic links in
        DIR after processing instances of '..'
        -P      use the physical directory structure without following symbolic
        links: resolve symbolic links in DIR before processing instances
```

help 还涉及 Bash 的其他主题，如果你感兴趣，可以不带命令行参数尝试 help 命令，看一下列出主题的清单。

一些命令行使用的新工具也经常缺少操作说明页面。这时候，最好的办法就是用 -h 或 --help 选项来调用 help 工具。例如：

```
jq --help

jq - commandline JSON processor [version 1.4]
Usage: jq [options] <jq filter> [file...]

For a description of the command line options and
how to write jq filters (and why you might want to)
see the jq manpage, or the online documentation at
http://stedolan.github.com/jq
```

指定 --help 选项对 cat 这样的 GNU 命令行工具也管用。不过，对应的操作说明页面通常会提供更多的信息。如果尝试了这三个方法，你仍然无计可施，则最好是向互联网求助。附录 A 中列出了书中使用的所有命令行工具，其中除了有命令行工具的安装指导，还有寻求帮助的方法。

2.4 延伸阅读

- Janssens, J. H. M. (2014). Data Science Toolbox. Retrieved May 10, 2014, from http://datasciencetoolbox.org.
- Oracle. (2014). VirtualBox. Retrieved May 10, 2014, from http://virtualbox.org.
- HashiCorp. (2014). Vagrant. Retrieved May 10, 2014, from http://vagrantup.com.
- Heddings, L. (2006). Keyboard Shortcuts for Bash. Retrieved May 10, 2014, from http://www.howtogeek.com/howto/ubuntu/keyboard-shortcuts-for-bash-command-shell-for-ubuntu-debian-suse-redhat-linux-etc.
- Peek, J., Powers, S., O'Reilly, T., & Loukides, M. (2002). *Unix Power Tools* (3rd Ed.). O'Reilly Media.

第 3 章

数据获取

本章介绍 OSEMN 模型的第一个步骤：数据获取。毕竟，没有数据就没有多少数据科学工作可做。这里我们假定，解决数据科学问题所需的数据已经以某种形式存在于某个地方。我们的目标是将这些数据以我们能处理的形式放入计算机（或数据科学工具箱）。

按照 Unix 哲学，文本是通用的接口。几乎所有的命令行工具都或者接受文本输入，或者产生文本输出，或者两者兼具。这就是多个命令行工具能够集聚一起协作的主要原因。不过，我们将会看到，即使是文本也可能有多种形式。

数据获取有多种途径，例如从服务器中下载、查询数据库，或者连接 Web API。有时数据是压缩形式或者二进制格式的（例如微软 Excel）。本章中，我们讨论能用命令行来帮助解决这些问题的几个工具，包括：`curl`（Stenberg，2012）、`in2csv`（Groskopf，2014）、`sql2csv`（Groskopf，2014）和 `tar`（Bailey，Eggert & Poznyakoff，2014）。

3.1 概述

本章将学习：

- 从互联网下载数据
- 查询数据库
- 连接 Web API
- 解压缩文件
- 将微软 Excel 电子表格转换为可用的数据

3.2 将本地文件复制到数据科学工具箱

一种常见的情况是你的计算机上已经有了必要的文件。本节介绍怎样将这些文件放入本地或远程的数据科学工具箱。

3.2.1 本地数据科学工具箱

在第 2 章我们曾经提到，本地数据科学工具箱是一个隔离的虚拟环境。幸好还有一个例外：可以将文件传入或传出数据科学工具箱。运行 `vagrant up` 时你所在的本地目录（也就是包含文件 Vagrantfile 的目录）会映射到数据科学工具箱的一个目录。该目录称为 /vagrant（注意这不是你的主目录）。让我们来看一下它的内容：

```
$ ls -1 /vagrant
Vagrantfile
```

如果你在本地计算机上有一个文件，并且想要用某些命令行工具对它进行处理，只需将这个文件复制或移动到该目录。假设你的计算机桌面上有一个名为 logs.csv 的文件。如果你运行的是 GUN/Linux 或 Mac OS X 系统，那就在操作系统（不是在数据科学工具箱）中进入包含 Vagrantfile 的目录，并执行下列命令：

```
$ cp ~/Desktop/logs.csv .
```

如果使用的是 Windows 系统，可以在命令提示符或 PowerShell 上运行下列命令：

```
> cd %UserProfile%\Desktop
> copy logs.csv MyDataScienceToolbox\
```

也可以使用 Windows 资源管理器将文件拖拽到该目录。

现在该文件就位于目录 /vagrant 中了。将数据放到一个单独的目录中（这里我们使用 ~/book/ch03/data）是一个好想法。因此，在复制文件之后，可以用下列命令来移动它：

```
$ mv /vagrant/logs.csv ~/book/ch03/data
```

3.2.2 远程数据科学工具箱

如果你运行的是 Linux 或 Mac OS X 系统，可以使用 `scp`（Rinne & Ylonen，2014），意为“安全复制”（secure copy），将文件复制到 EC2 实例上。这时需要一个密钥对文件，就是你登录运行着数据科学工具箱的 EC2 实例时用到的密钥对文件：

```
$ scp -i mykey.pem ~/Desktop/logs.csv \
> ubuntu@ec2-184-73-72-150.compute-1.amazonaws.com:data
```

将例子中的主机名（`@` 和 `:` 之间的部分）替换为你在 AWS 控制台的 EC2 概述页面上看到的值。

3.3 解压缩文件

如果原始的数据集非常巨大，或者包含了许多文件，那么这个文件可能是（压缩的）归档文件。包含许多重复值（例如文本文件中的单词或 JSON 文件中的 key）的数据集特别适合压缩。

压缩归档文件的常见扩展名是：.tar.gz、.zip 和 .rar。如果要解压缩，应分别使用命令行工具 `tar`（Bailey，Eggert & Poznyakoff，2014）、`unzip`（Smith，2009）和 `unrar`（Asselstine，Scheurer & Winkelmann，2014）。另外，还有一些不太常见的文件扩展名，需要使用其他工具来解压缩。举例来说，提取名为 logs.tar.gz 的文件，应使用：

```
$ cd ~/book/ch03
$ tar -xzvf data/logs.tar.gz
```

`tar` 的繁琐的选项确实令人头疼。上面的例子中，四个选项 `x`、`z`、`v` 和 `f` 指定了 `tar` 应从归档文件中提取数据，使用 gzip 作为解压算法，使用详细模式（verbose），使用文件 logs.tar.gz。假以时日，你会习惯于输入这四个选项字符，但其实还有更加简便的方式。

使用称为 `unpack`（Brisbin，2013）的便捷脚本，无需记住不同的命令行工具及其选项，就可以解压缩许多不同格式的文件。`unpack` 通过查看要解压缩的文件扩展名，调用相应的命令行工具。

`unpack` 工具包含在了数据科学工具箱中。请记住可以在附录 A 中查阅它的安装方法。示例 3-1 展示了 `unpack` 的源代码。虽然 Bash 脚本编程并不是本书关注的焦点，但是花一点时间理解它是如何运行的还是有用的。

示例 3-1　解压缩各种文件格式（`unpack`）

```
#!/usr/bin/env bash
# unpack: Extract common file formats

# Display usage if no parameters given
  if [[ -z "$@" ]]; then
    echo " ${0##*/} <archive> - extract common file formats)"
    exit
  fi

# Required program(s)
  req_progs=(7z unrar unzip)
  for p in ${req_progs[@]}; do
    hash "$p" 2>&- || \
    { echo >&2 " Required program \"$p\" not installed."; exit 1; }
  done

# Test if file exists
  if [ ! -f "$@" ]; then
    echo "File "$@" doesn't exist"
```

```
    exit
  fi

# Extract file by using extension as reference
  case "$@" in
    *.7z ) 7z x "$@" ;;
    *.tar.bz2 ) tar xvjf "$@" ;;
    *.bz2 ) bunzip2 "$@" ;;
    *.deb ) ar vx "$@" ;;
    *.tar.gz ) tar xvf "$@" ;;
    *.gz ) gunzip "$@" ;;
    *.tar ) tar xvf "$@" ;;
    *.tbz2 ) tar xvjf "$@" ;;
    *.tar.xz ) tar xvf "$@" ;;
    *.tgz ) tar xvzf "$@" ;;
    *.rar ) unrar x "$@" ;;
    *.zip ) unzip "$@" ;;
    *.Z ) uncompress "$@" ;;
    * ) echo " Unsupported file format" ;;
  esac
```

现在，对同样的文件解压缩，只需要使用：

```
$ unpack logs.tar.gz
```

3.4 微软Excel电子表格的转换

对许多人来说，微软 Excel 提供了一种直观地处理和计算小规模数据集的方式。因此，大量数据都以微软 Excel 电子表格的形式呈现。根据其文件扩展名的不同，这些电子表格存储在专有的二进制格式文件（.xls）或压缩 XML 的文件集合（.xlsx）中。这两种情况下，多数命令行工具都不能直接使用这些数据。如果仅仅因为数据以这种方式存储，就无法使用这些宝贵的数据集，那就太遗憾了。

幸好有一个称为 `in2csv`（Groskopf，2014）的命令行工具，能够将微软 Excel 电子表格转换为 CSV 文件。CSV 代表用逗号分隔开的值或字符分隔开的值。使用 CSV 可能比较棘手，因为它缺少形式化规范。RFC 4180（http://www.ietf.org/rfc/rfc4180.txt）基于以下三点定义 CSV 格式。

(1) 每条记录占据独立一行，由换行符（CRLF）分隔开。例如：

```
aaa,bbb,ccc CRLF
zzz,yyy,xxx CRLF
```

(2) 文件中的最后一条记录有无末尾换行符皆可。例如：

```
aaa,bbb,ccc CRLF
zzz,yyy,xxx
```

(3) 在文件的首行可以有标题行（可选的），其格式与普通记录行相同。标题行要包含文件中对应字段的名字，字段数量还应与文件中其他记录相同（是否存在标题行应由该文件的 MIME 类型的可选的 header 参数来指定）。例如：

```
field_name,field_name,field_name CRLF
aaa,bbb,ccc CRLF
zzz,yyy,xxx CRLF
```

让我们使用由互联网电影数据库（Internet Movie Database，IMDb）中的前 250 部电影构成的电子表格来说明 in2csv。这个文件名为 imdb-250.x.sx，可从 http://www.overthinkingit.com/2011/10/11/imdb-top-250-movies-4th-edition/2/ 下载。使用如下方式调用 in2csv，抽取其中的数据：

```
$ cd ~/book/ch03
$ in2csv data/imdb-250.xlsx > data/imdb-250.csv
```

这种情况下，扩展名 .xlsx 自动确定了文件的格式。如果我们要用管道把数据输入 in2csv，就必须显式地指定格式。让我们看一下数据：

```
$ in2csv data/imdb-250.xlsx | head | cut -c1-80
Title,title trim,Year,Rank,Rank (desc),Rating,New in 2011 from 2010?,2010 rank,R
Sherlock Jr. (1924),SherlockJr.(1924),1924,221,30,8,y,n/a,n/a,
The Passion of Joan of Arc (1928),ThePassionofJoanofArc(1928),1928,212,39,8,y,n/
His Girl Friday (1940),HisGirlFriday(1940),1940,250,1,8,y,n/a,n/a,
Tokyo Story (1953),TokyoStory(1953),1953,248,3,8,y,n/a,n/a,
The Man Who Shot Liberty Valance (1962),TheManWhoShotLibertyValance(1962),1962,2
Persona (1966),Persona(1966),1966,200,51,8,y,n/a,n/a,
Stalker (1979),Stalker(1979),1979,243,8,8,y,n/a,n/a,
Fanny and Alexander (1982),FannyandAlexander(1982),1982,210,41,8,y,n/a,n/a,
Beauty and the Beast (1991),BeautyandtheBeast(1991),1991,249,2,8,y,n/a,n/a,
```

如你所见，默认情况下 CSV 是不易阅读的。可以通过管道将数据输入称为 csvlook 的工具（Groskopf，2014），它会令人满意地将数据格式化为表格。这里，为了使表格适合页面宽度，我们使用 csvcut 来显示一部分列。

```
$ in2csv data/imdb-250.xlsx | head | csvcut -c Title,Year,Rating | csvlook
|-----------------------------------------+------+---------|
| Title                                   | Year | Rating  |
|-----------------------------------------+------+---------|
| Sherlock Jr. (1924)                     | 1924 | 8       |
| The Passion of Joan of Arc (1928)       | 1928 | 8       |
| His Girl Friday (1940)                  | 1940 | 8       |
| Tokyo Story (1953)                      | 1953 | 8       |
| The Man Who Shot Liberty Valance (1962) | 1962 | 8       |
| Persona (1966)                          | 1966 | 8       |
| Stalker (1979)                          | 1979 | 8       |
| Fanny and Alexander (1982)              | 1982 | 8       |
| Beauty and the Beast (1991)             | 1991 | 8       |
|-----------------------------------------+------+---------|
```

电子表格可能包含多个工作表。in2csv 默认提取第一张工作表。要提取另一张工作表，需要将表名传递给 --sheet 选项。

工具 in2csv、csvcut 和 csvlook 实际上都是 Csvkit 的成员，Csvkit 是处理 CSV 数据的命令行工具集。它包含了许多重要的工具，在本书中将经常用到。如果你正在使用数据科学工具箱，那就已经安装了 Csvkit。否则，请参考附录 A 的说明来安装它。



in2csv 之外的另一种方法是，用微软 Excel 或开源电子表格程序（例如 LibreOffice Calc）打开电子表格，然后手工将其导出为 CSV 文件。这是一次性的解决方法，其缺点是不易扩展到多个文件，也不能自动化。更何况，当你使用远程服务器上的命令行时，很可能找不到这样的应用程序。

3.5　查询关系数据库

大多数公司将数据存储在关系数据库中。关系数据库的例子包括 MySQL、PostgreSQL 和 SQLite。这些数据库各有着些许不同的交互方式。有的提供命令行工具或命令行接口，有的则不提供。此外，它们的用法和输出也不完全一致。

幸好 Csvkit 套件中有一个称为 sql2csv 的命令行工具。由于它利用了 Python SQLAlchemy 软件包，只需要使用一个工具就可以在许多不同的数据库上执行查询，包括 MySQL、Oracle、PostgreSQL、SQLite、微软 SQL Server 和 Sybase。正如它的名字所暗示的，sql2csv 的输出是 CSV 格式。

可以通过在关系数据库上执行 SELECT 查询来获取数据。（sql2csv 也支持 INSERT、UPDATE 和 DELETE 查询，但这不属于本章的范围。）要从名为 iris.db 的 SQLite 数据库中选取特定的数据集，可以用如下方式调用 sql2csv：

```
$ sql2csv --db 'sqlite:///data/iris.db' --query 'SELECT * FROM iris '\
> 'WHERE sepal_length > 7.5'
sepal_length,sepal_width,petal_length,petal_width,species
7.6,3.0,6.6,2.1,Iris-virginica
7.7,3.8,6.7,2.2,Iris-virginica
7.7,2.6,6.9,2.3,Iris-virginica
7.7,2.8,6.7,2.0,Iris-virginica
7.9,3.8,6.4,2.0,Iris-virginica
7.7,3.0,6.1,2.3,Iris-virginica
```

这里，我们选择了 sepal_length 大于 7.5 的所有行。--db 选项指定了数据库的 URL，其典型形式是：dialect+driver://username:password@host:port/database。

3.6 从互联网下载

毫无疑问，互联网提供了最大的数据资源。这些数据呈现不同的形式，使用不同的协议。命令行工具 cURL（Stenberg，2012）可被视为用命令行从互联网下载数据的“瑞士军刀”。

通过浏览器访问 URL（Uniform Resource Locator，统一资源定位器）时，所下载的数据可以得到解释。例如，HTML 文件呈现为网站，MP3 文件可以自动播放，PDF 文件可以由阅读器自动打开。但是，当用 cURL 来访问 URL 时，数据被原封不动地下载，然后交给标准输出显示出来，最后可以由其他命令行工具来做进一步处理。

调用 `curl` 最简单的方法是仅指定一个 URL 作为命令行的参数。例如，要从古腾堡计划（Project Gutenberg）中下载马克·吐温的《哈克贝利·费恩历险记》，可以运行如下命令：

```
$ curl -s http://www.gutenberg.org/cache/epub/76/pg76.txt | head -n 10

The Project Gutenberg EBook of Adventures of Huckleberry Finn, Complete
by Mark Twain (Samuel Clemens)
This eBook is for the use of anyone anywhere at no cost and with almost
no restrictions whatsoever. You may copy it, give it away or re-use
it under the terms of the Project Gutenberg License included with this
eBook or online at www.gutenberg.net
```

默认情况下，`curl` 输出一个进度条，显示下载速率和预期的完成时间。如果你将输出直接通过管道连接到 `head` 这样的其他命令行工具，切记要指定 `-s`（代表 silent）选项，这样就不会显示进度条。以下列命令为例，将其输出与上面的输出作比较：

```
$ curl http://www.gutenberg.org/cache/epub/76/pg76.txt | head -n 10
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0       0 --:--:-- --:--:-- --:--:--

The Project Gutenberg EBook of Adventures of Huckleberry Finn, Complete
by Mark Twain (Samuel Clemens)

This eBook is for the use of anyone anywhere at no cost and with almost
no restrictions whatsoever. You may copy it, give it away or re-use
it under the terms of the Project Gutenberg License included with this
eBook or online at www.gutenberg.net
```

你可以注意到，第二条命令的输出中没有关闭进度条，因此包含了多余的文本甚至错误消息。如果将这样的数据保存到文件中，那就不必指定 `-s` 选项：

```
$ curl http://www.gutenberg.org/cache/epub/76/pg76.txt > data/finn.txt
```

也可以用 `-o` 选项明确地指定输出文件来保存数据：

```
$ curl -s http://www.gutenberg.org/cache/epub/76/pg76.txt -o data/finn.txt
```

从互联网下载数据时，对应的 URL 最可能使用 HTTP 或 HTTPS 协议。如果要从 FTP（文件传输协议）服务器上下载数据，用相同的方式使用 `curl` 即可。当 URL 有密码保护时，可以用如下方式指定用户名和密码：

```
$ curl -u username:password ftp://host/file
```

如果该 URL 是一个目录，`curl` 会将目录内容列出来。

如果访问的是简写的 URL，例如以 http://bit/ly/ 或 http://t.co/ 作为前缀的 URL，浏览器会自动重定向到正确的地址。不过，如果使用 `curl`，就需要指定 `-L` 或 `--location` 选项来重定向：

```
$ curl -L j.mp/locatbbar
```

如果没有指定 `-L` 或 `--location` 选项，可能会得到如下信息：

```
$ curl j.mp/locatbbar
<html>
<head>
<title>bit.ly</title>
</head>
<body>
<a href="http://en.wikipedia.org/wiki/List_of_countries_and_territories_by_bo
rder/area_ratio">moved here</a>
</body>
```

如果指定 `-I` 或 `--head` 选项，`curl` 就只获取响应信息的 HTTP 头部数据：

```
$ curl -I j.mp/locatbbar
HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Wed, 21 May 2014 18:50:28 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Cache-Control: private; max-age=90
Content-Length: 175
Location: http://en.wikipedia.org/wiki/List_of_countries_and_territories_by_bo
Mime-Version: 1.0
Set-Cookie: _bit=537cf574-002ba-07d79-2e1cf10a;domain=.j.mp;expires=Mon Nov 17
```

第一行指出了 HTTP 状态码，这里是 301（永久移除）。你也可以看到该 URL 重定向的地 址：http://en.wikipedia.org/wiki/List_of_countries_and_territories_by_border/area_ratio。 在 `curl` 没有得出期望的结果时，检查头部数据并获取状态码是一个有用的调试工具。其他常见的 HTTP 状态码包括 404（未找到）和 403（被禁止）。（完整的 HTTP 状态码清单，参见维基百科：http://en.wikipedia.org/wiki/List_of_HTTP_status_codes。）

总结本节的内容，cURL 是从互联网下载数据的简单直接的命令行工具。它最常见的三个选项分别是：用来取消进度条的 `-s`、指定用户名和密码的 `-u`，以及自动跟踪重定向的 `-L`。更多信息请参见其操作说明页面。

3.7　调用Web API

上一节我们解释了怎样从互联网下载单个文件。从互联网中获取数据的另一种方法是通过 Web API（Application Programming Interface，应用程序编程接口）。各种组织所提供的 API 的数量一直在以不断加快的速度增长，这意味着有大量有趣的数据供我们数据科学家使用。

Web API 并不会像网站那样以良好的布局呈现出来，而是大多以 JSON 或 XML 这样的结构化形式返回数据。结构化形式的数据的优势在于易于被其他工具（比如 `jq`）处理。例如，http://randomuser.me 的 API 以下列 JSON 结构返回数据：

```
$ curl -s http://api.randomuser.me | jq '.'
{
  "results": [
    {
      "version": "0.3.2",
      "seed": "1c5b868416387bf",
      "user": {
        "picture": "http://api.randomuser.me/0.3.2/portraits/women/2.jpg",
        "SSN": "972-79-4140",
        "cell": "(519)-135-8132",
        "phone": "(842)-322-2703",
        "dob": "64945368",
        "registered": "1136430654",
        "sha1": "a3fed7d4f481fbd6845c0c5a19e4f1113cc977ed",
        "gender": "female",
        "name": {
          "last": "green",
          "first": "scarlett",
          "title": "miss"
        },
        "location": {
          "zip": "43413",
          "state": "nevada",
          "city": "redding",
          "street": "8608 crescent canyon st"
        },
        "email": "scarlett.green32@example.com",
        "username": "reddog82",
        "password": "ddddd",
        "salt": "AEKvMi$+",
        "md5": "f898fc73430cff8327b91ef6d538be5b"
      }
    }
  ]
}
```

数据通过管道进入 `jq` 命令行工具，从而清晰地显示出来。`jq` 还有许多其他功能，将在第 5 章深入探讨。

有的 Web API 以流的形式返回数据。这意味着一旦连接了 API，数据将源源不断地到达。

一个著名的例子是 Twitter firehose，它持续不断地返回全世界发送的推文（tweet）流。幸好我们使用的大多数命令行工具也以流的方式运行，因此我们也可以使用这类数据。

有的 API 需要使用 OAuth 协议登录。有一个称为 `curlicue`（Foster，2014）的简便的命令行工具，可以帮助进行所谓的 OAuth 握手（OAuth dance）。一旦握手启动，`curlicue` 就以正确的头部数据来调用 `curl`。首先，你要用 `curlicue-setup` 对特定的 API 获取授权，然后就可以用 `curlicue` 调用该 API。例如，要使用 `curlicue` 从 Twitter API 获取数据，应该这样做：

```
$ curlicue-setup \
> 'https://api.twitter.com/oauth/request_token' \
> 'https://api.twitter.com/oauth/authorize?oauth_token=$oauth_token' \
> 'https://api.twitter.com/oauth/access_token' \
> credentials
$ curlicue -f credentials \
> 'https://api.twitter.com/1/statuses/home_timeline.xml'
```

对付更加流行的 API，有专门的命令行工具。它们都是包装器（wrapper），能以简便的方式连接 API。在第 9 章中，我们将使用命令行工具 `bigmler` 来连接 BigMl 的预测 API。

3.8 延伸阅读

- Molinaro, A. (2005). *SQL Cookbook*. O’Reilly Media.
- Wikipedia. (2014). List of HTTP status codes. Retrieved May 10, 2014, from http://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

第 4 章
创建可重用的命令行工具

本书中使用了许多基本上可以在一行内写下的命令和管道（我们称之为单行，one-liner）。仅使用单行就可以执行复杂任务是命令行之所以强大的原因所在。与编写传统计算机程序相比，使用单行是迥然不同的体验。

有的任务只要执行一次，有的则要执行多次。有的任务十分特殊，有的则很普遍。如果你预料到或注意到自己需要经常重复使用某个单行，那就值得将它变为自己的命令行工具。单行和命令行工具都有各自的用武之地。要想抓住机会需要实践和技巧。命令行工具的优势在于不需要你记住整个单行，并且如果将它加入其他管道能够提高可读性。

使用编程语言的好处在于程序代码是存储在文件中的。这意味着这些代码易于重用。如果代码带有参数，甚至可以将其用于具有相似模式的问题中。

拥有命令行工具真的是两全其美：既可以在命令行和接受参数中使用，又只需要创建一次。本章中，我们将用两种方式来熟悉怎样创建可重用的命令行工具。首先，解释怎样将单行转变为可重用的命令行工具。通过向命令中加入参数，可以使其具备与程序语言相同的灵活性。然后，演示怎样从已经用程序语言编写完毕的代码开始，创建可重用的命令行工具。按照 Unix 哲学，你的代码可以与其他命令行工具结合起来，这些工具可能是用完全不同的程序语言编写的。这里，我们将关注两种程序语言：Python 和 R。

我们相信，长远来看，创建可重用的工具会使你成为更加高效和多产的数据科学家。你要逐步创建自己的数据科学工具箱，以便从中提取现有的工具，将其应用到与以前遇到的问题类似的新问题中。你需要实践，才有能力抓住机会将单行或已有代码转变为命令行工具。

要将单行转变为 shell 脚本，需要使用一些 shell 脚本语言。我们只演示 shell 脚本语言中一小部分概念的用处。完整的 shell 脚本语言课程需要专门的书籍来讲解，这超出了本书的范围。如果想要深入研究，推荐你阅读 Robbins & Beebe（2005）的 *Classic Shell Scripting*（http://shop.oreilly.com/product/9780596005955.do）一书。

4.1 概述

本章你将学习：

- 将单行转变为 shell 脚本
- 使已有的 Python 和 R 代码成为命令行的一部分

4.2 将单行转变为shell脚本

本节将解释怎样将单行转变为可重用的命令行工具。假设有如下单行：

```
$ curl -s http://www.gutenberg.org/cache/epub/76/pg76.txt | ❶
> tr '[:upper:]' '[:lower:]' | ❷
> grep -oE '\w+' |           ❸
> sort |                     ❹
> uniq -c |                  ❺
> sort -nr |                 ❻
> head -n 10                 ❼
  6441 and
  5082 the
  3666 i
  3258 a
  3022 to
  2567 it
  2086 t
  2044 was
  1847 he
  1778 of
```

简言之，你可能从输出中猜到，这个单行返回 *Adventures of Huckleberry Finn* 电子书的前 10 个单词。它是通过下列步骤完成的：

❶ 用 `curl` 下载电子书。

❷ 用 `tr`（Meyering，2012）将整个文本转换为小写。

❸ 用 `grep`（Meyering，2012）提取所有的单词，每个单词占一行。

❹ 用 `sort`（Haertel & Eggert，2012）将全部单词按字母顺序排列。

❺ 用 `uniq`（Stallman & MacKenzie，2012）删除重复的单词，并计算每个单词在列表中出现的次数。

❻ 用 `sort` 按单词出现次数的降序，对这个无重复单词的列表排序。

❼ 用 `head` 取前 10 行（也即单词）。



这个单行中使用的每个命令行工具都有操作说明页。因此，当你想要知道更多信息时，例如 grep，就可以在命令行中运行 man grep。下一章将更详细地介绍命令行工具 tr、grep、uniq 和 sort。

只将单行运行一次无可厚非。不过，试想，如果我们想要找到古腾堡计划所有电子书的前 10 个单词，或者设想我们想找到某新闻网站以小时为单位的前 10 个单词。这些情况下，最好是将这个单行作为独立的构件来组成更大的东西。我们想要以参数的形式给单行加入一些灵活性，因此将其转换为 shell 脚本。

由于我们使用 Bash 作为 shell，编写脚本的程序语言就是 Bash。这允许我们将单行作为起点，逐步对其加以改进。我们将带你通过下列 6 个步骤，将单行变为可重用的命令行工具。

(1) 复制并粘贴单行到文件中。
(2) 添加执行权限。
(3) 定义所谓的 shebang。
(4) 删除固定的输入部分。
(5) 添加参数。
(6) 视情况扩展 PATH。

4.2.1 步骤1：复制和粘贴

第一个步骤是创建一个新文件。打开你最中意的文本编辑器，复制和粘贴单行。将该文件命名为 top-words-1.sh（1 代表创建新命令行工具的第一个步骤），将其放入 ~/book/ch04 目录中，也可以选择其他名称和存放目录。文件的内容如示例 4-1 所示。

示例 4-1 ~/book/ch04/top-words-1.sh

```
curl -s http://www.gutenberg.org/cache/epub/76/pg76.txt |
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n 10
```

我们使用文件扩展名 .sh 来表明创建的是一个 shell 脚本。不过，命令行工具并不需要扩展名。实际上，命令行工具很少有扩展名。



这里有一个关于命令行的小窍门。在命令行中，!! 会被替换为刚才运行的命令。因此，如果你意识到先前的命令需要超级用户权限，可以运行 sudo !!（Miller，2013）。此外，如果想要不通过复制和粘贴就将先前的命令保存到文件中，可以运行 echo " !! " > scriptname。在运行命令之前，一定要检查 scriptname 文件的内容是否正确，因为如果命令中有引号，有时会失效。

现在可以使用命令行工具 bash（Fox & Ramey，2010）来解释和执行文件中的命令：

```
$ bash ~/book/ch04/top-words-1.sh
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
   2567 it
   2086 t
   2044 was
   1847 he
   1778 of
```

第一个步骤已经使我们下次使用时不用再输入单行。由于文件无法单独被执行，还不能称作真正的命令行工具。让我们在下一个步骤中改变这种情况。

4.2.2　步骤2：添加执行权限

无法直接执行该文件的原因在于我们没有正确的访问权限。具体地说，作为用户的你需要有执行该文件的权限。本节我们将改变文件的访问权限。



为了显示步骤之间的差异，我们使用 cp top-words-{1, 2}.sh 将文件复制到 top-words-2.sh。你也可以一直使用同一个文件。

要改变文件的访问权限，我们使用称为 chmod（MacKenzie & Meyering，2012）的命令行工具，意为“改变模式”（change mode）。它改变特定文件的文件模式位。下列命令授予用户（就是你）执行 top-words-2.sh 的权限：

```
$ cd ~/book/ch04/
$ chmod u+x top-words-2.sh
```

u+x 选项包括三个特性：(1) u 表明想要改变文件拥有者（就是你）的权限，因为是你创建了该文件；(2) + 表明想要添加权限；(3) x 表明是执行权限。现在来看一下这两个文件的权限：

```
$ ls -l top-words-{1,2}.sh
-rw-rw-r-- 1 vagrant vagrant 145 Jul 20 23:33 top-words-1.sh
-rwxrw-r-- 1 vagrant vagrant 143 Jul 20 23:34 top-words-2.sh
```

第一列显示的是每个文件的访问权限。对于 top-words-2.sh，是 -rwxrw-r--。第一个字符，-，表明文件类型。- 意为常规文件，d（这里没有出现）意为目录。随后的三个字符，rwx，表明文件拥有者的访问权限。r 和 w 分别代表读和写。（你可以看到，top-words-q.sh

用 - 替换了 x，意思是我们不能执行该文件。）后面的三个字符，rw-，表明拥有该文件的用户组中所有成员的访问权限。最后，第一列的最后三个字符，r--，表明其他所有用户的访问权限。

现在就可以执行该文件：

```
$ ~/book/ch04/top-words-2.sh
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
   2567 it
   2086 t
   2044 was
   1847 he
   1778 of
```

注意，如果你所在的目录与可执行文件的目录相同，需要用如下方式执行该文件（注意其中的 ./）：

```
$ cd ~/book/ch04
$ ./top-words-2.sh
```

如果尝试执行你没有正确访问权限的文件，比如 top-words-1.sh，你会看到如下报错信息：

```
$ ./top-words-1.sh
bash: ./top-words-1.sh: Permission denied
```

4.2.3　步骤3：定义shebang

尽管已经可以单独执行文件本身，我们还应该向文件中加入所谓的 shebang。shebang 是脚本中的特殊行，它告诉系统用哪个可执行文件来解释命令。在我们的例子中，想要使用 bash 来解释命令。示例 4-2 显示了文件 top-words-3.sh 加入 shebang 的样子。

示例 4-2　~/book/ch04/top-words-3.sh

```
#!/usr/bin/env bash
curl -s http://www.gutenberg.org/cache/epub/76/pg76.txt |
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n 10
```

shebang 这个名字源于该行的前两个字符：井号 #（she）和惊叹号 !（bang）。像我们在前面几个步骤中那样把它省略掉不是好主意，因为那样的话脚本的行为就没有定义。我们所使用的 Bash shell 默认使用 /bin/bash 这个可执行文件。其他 shell 可能有不同的默认值。



有时你会碰到脚本中的 shebang 是 `!/usr/bin/bash` 或 `!/usr/bin/python`（在使用 Python 的情况下，下一节会讲到）形式。一般情况下这没问题，但如果 `bash` 或 `python`（Python Software Foundation，2014）可执行文件的安装目录并非 `/usr/bin`，那么这个脚本就再也无法执行。最好是使用我们这里的方式，也就是 `!/usr/bin/env bash` 和 `!/usr/bin/env python`，这是因为 `env`（Mlynarik & MacKenzie，2012）命令行工具能够获知 `bash` 和 `python` 的安装目录。简言之，使用 `env` 会使你的脚本可移植性更好。

4.2.4　步骤4：删除固定的输入

现在，我们有了一个可以在命令行中执行的有效的命令行工具。但我们还可以做得更好一些，即让命令行工具更具可重用性。我们文件中的第一个命令是 `curl`，它下载一个文本，从中我们期望获得前 10 个使用最多的单词。可见，数据和操作都是捆绑在一起的。

但是，如果我们想要从另外的电子书或其他任意的文本中获取前 10 个使用最多的单词会怎样？输入数据原本是固定于工具本身之中的，将数据从命令行工具中分离出来会更好。

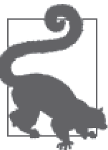
如果假设命令行工具的用户将会提供文本，那就更加普遍适用。因此，解决方法是简单地将 `curl` 命令从脚本中删除。更新后的脚本命名为 top-words-4.sh，如示例 4-3 所示。

示例 4-3　~/book/ch04/top-words-4.sh

```
#!/usr/bin/env bash
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n 10
```

这样是可以的，因为如果某个脚本的第一条命令（如 `tr`）需要来自标准输入的数据，那么它将获得给命令行工具的输入。假设我们已经将电子书保存到 data/finn.txt，举例来说，可以这样做：

```
$ cat data/ | ./top-words-4.sh
```



尽管我们还没有在脚本里这样做，但同样的原理也适用于保存数据。一般来说，让用户来处理输入和输出更好一些。当然，如果你只打算在自己的项目中使用命令行工具，那脚本写得多么具体都无所谓。

4.2.5　步骤5：参数化

为了使我们的命令行工具更加可重用，还可以再执行一个步骤：参数化。在我们的命令行工具中，有许多固定的命令行参数，例如，`sort` 命令的 `-nr` 以及 `head` 命令的 `-n 10`。让前一个参数保持不变很可能是最好的。但对 `head` 命令来说，允许有不同的参数值会很有用。这使得终端用户可以设置要输出的最常使用单词的数量。示例 4-4 显示了将 `head` 参数化后

文件 top-words-5.sh 的样子。

示例 4-4 ~/book/ch04/top-words-5.sh

```
#!/usr/bin/env bash
NUM_WORDS="$1"                                          ❶
tr '[:upper:]' '[:lower:]' | grep -oE '\w+' | sort |
uniq -c | sort -nr | head -n $NUM_WORDS                 ❷
```

❶ 变量 NUM_WORDS 设置为值 $1，这是 Bash 中的一个特殊值。它存放的是传递给命令行工具的第一个命令行参数的值。

❷ 注意，为了使用 NUM_WORDS 变量的值，需要在它前面加上美元符号（$）。当要设置它时，就不要加美元符号。



可以直接将 $1 用作 head 命令 -n 选项的值，不必繁琐地创建 NUM_WORDS 这样的额外变量。不过，对于较大的脚本，以及更多的命令行参数，如 $2 和 $3，使用命名的变量会让代码的可读性更高。

现在，如果想要看一下文本中前 5 个使用最多的单词，我们将这样调用命令行工具：

```
$ cat data/finn.txt | top-words-5.sh 5
```

如果用户没有提供参数，head 将返回报错信息，因为 $1 的值以及 NUM_WORDS 都会是空字符串。

```
$ cat data/finn.txt | top-words-5.sh
head: option requires an argument -- 'n'
Try 'head --help' for more information.
```

4.2.6 步骤6：扩展PATH

现在终于创建完了一个可重用的命令行工具。不过，还有一个步骤可能很有用。这个可选步骤确保你可以在任何地方执行你的命令行工具。

现在，要执行命令行工具，你要么进入工具所在的目录，要么像步骤 2 那样包含完整的路径名。如果命令行工具是为特定项目专门创建的，这无可厚非。但是，如果你的命令行工具可能应用于多个场合，那么像已经安装的命令行工具那样，在任何地方都能执行就很有用。

要达到这个目标，Bash 需要知道到哪里寻找你的命令行工具。为此它浏览一个目录列表，这个列表存储在称为 PATH 的环境变量中。在新安装的数据科学工具箱中，PATH 的内容是：

```
$ echo $PATH | fold
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/home/vagrant/tools:/usr/lib/go/bin:/home/vagrant/.go/bin:/home/vagrant
/.data-science-at-the-command-line/tools:/home/vagrant/.bin
```

目录由冒号间隔。下面是目录列表：

```
$ echo $PATH | tr : '\n' | sort
/bin
/home/vagrant/.bin
/home/vagrant/.data-science-at-the-command-line/tools
/home/vagrant/.go/bin
/home/vagrant/tools
/sbin
/usr/bin
/usr/games
/usr/lib/go/bin
/usr/local/bin
/usr/local/games
/usr/local/sbin
/usr/sbin
```

要永久修改 `PATH`，需要编辑主目录中的 .bashrc 或 .profile 文件。如果你将定制的全部命令行工具放入一个目录，如 ~/tools，那么只需要修改 `PATH` 一次。你会看到，数据科学工具箱已经在其 `PATH` 中包含了 /home/vagrant/.bin。现在你不再需要加入 `./`，但是可以只使用文件名。而且，你不再需要记住命令行工具的位置，因为可以使用 `which` 命令来定位。

4.3　用Python和R创建命令行工具

上一节我们创建的命令行工具是用 Bash 编写的。（当然，我们并没有利用 Bash 语言的所有特性，不过解释器还是 `bash`。）现在你可能知道，命令行是与语言无关的，因此并不是必须使用 Bash 来创建命令行工具。

这一节你将看到，也可以用其他程序语言来创建命令行工具。我们主要关注 Python 和 R，因为它们是当前数据科学界中最流行的程序语言。对这两种语言的完整介绍超出了本书的范围，因此我们假设你已经在一定程度上熟悉了 Python 和 R。如果用 Java、Go 和 Julia 等程序语言来创建命令行工具，也遵循相同的模式。

不用 Bash 而用其他程序语言来创建命令行工具有三个主要的原因。首先，你可能已经有一些代码，希望能在命令行中使用它们。其次，命令行工具可能封装一百多行代码。最后，命令行工具的速度要很快。

上一节的 6 个步骤大致也适用于用其他程序语言创建命令行工具。不过第一步不是在命令行中复制和粘贴，而应将相关的代码复制并粘贴到新文件中。用 Python 和 R 的命令行工具需要在 shebang 后面分别指定 `python`（Python Software Foundation，2014）和 `Rscript`（R Foundation for Statistical Computing，2014）作为解释器。

使用 Python 和 R 创建命令行工具时，有两个方面的问题要特别注意，下面会详细讨论。首先，对标准输入的处理，在 shell 脚本中是自然而然的，在 Python 和 R 中则要显式地处

理。其次，由于用 Python 和 R 编写的命令行工具通常更复杂，可能还得让用户能够指定更复杂的命令行参数。

4.3.1　移植shell脚本

首先让我们看一下怎样将以前的 shell 脚本移植到 Python 和 R 中。换句话说，Python 和 R 代码计算的来自标准输入的最常使用单词是什么样？用 shell 之外的程序语言实现这个任务是否是一个好主意并不重要。真正重要的是它给我们一个好机会来比较 Bash 与 Python 和 R。

我们首先展示 top-words.py 和 top-words.R 两个文件，然后讨论它们与 shell 代码的差别。在 Python 中，代码可能如示例 4-5 所示。

示例 4-5　~/book/ch04/top-words.py

```
#!/usr/bin/env python
import re
import sys
from collections import Counter
num_words = int(sys.argv[1])
text = sys.stdin.read().lower()
words = re.split('\W+', text)
cnt = Counter(words)
for word, count in cnt.most_common(num_words):
    print "%7d %s" % (count, word)
```



示例 4-5 使用的是纯 Python。如果要进行高级的文本处理，建议查一下 NLTK 程序包（Pekins，2010）。如果想要处理大量数值型数据，建议使用 Pandas 程序包（McKinney，2012）。

在 R 中，代码可能如示例 4-6 所示（这里要感谢 Hadley Wickham）。

示例 4-6　~/book/ch04/top-words.R

```
#!/usr/bin/env Rscript
n <- as.integer(commandArgs(trailingOnly = TRUE))
f <- file("stdin")
lines <- readLines(f)
words <- tolower(unlist(strsplit(lines, "\\W+")))
counts <- sort(table(words), decreasing = TRUE)
counts_n <- counts[1:n]
cat(sprintf("%7d %s\n", counts_n, names(counts_n)), sep = "")
close(f)
```

让我们看一下三种实现（即 Bash、Python 和 R）返回的前 5 个单词及其数量是否相同：

```
$ < data/76.txt ./top-words-5.sh 5
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
$ < data/76.txt ./top-words.py 5
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
$ < data/76.txt ./top-words.R 5
   6441 and
   5082 the
   3666 i
   3258 a
   3022 to
```

非常好！当然，输出本身并没有多么令人兴奋。真正令人兴奋的是我们可以用多种方法完成同一个任务。现在，让我们看一下这些方法之间的差别。

首先，最显而易见的就是代码量的差异。对这个特定任务来说，Python 和 R 的代码量都比 Bash 多许多。这也说明，对某些任务来说，使用命令行效率更高。对其他任务来说，使用程序语言则更好。随着使用命令行的经验愈加丰富，你会开始意识到应该在什么时候使用哪种方法。当所有代码都转变为命令行工具时，你甚至可以将任务分解为子任务，并将 Bash 命令行工具与 Python 命令行工具等结合起来。不管用哪种方法完成任务最好，都可以信手拈来！

4.3.2　处理来自标准输入的流数据

在前面两个代码示例中，Python 和 R 都一次性读取整个标准输入。在命令行中，多数命令行工具以流的方式将数据通过管道传递给下一个命令行工具。[有一些命令行工具需要取得全部数据，才能将数据写到标准输出，如 `sort` 和 `awk`（Brennan，1994）。] 这意味着管道被这类命令行工具阻塞了。当输入数据有限的时候（如文件），这并不是什么问题。但是，当输入数据是持续不断的数据流时，这样的阻塞命令行工具就没有用了。

幸运的是，Python 和 R 都能处理流数据。例如，你可以逐行地应用函数。示例 4-7 和示例 4-8 是两个很小的例子，分别解释了 Python 和 R 是怎样做的。它们计算通过管道传递过来的所有整数的平方。

示例 4-7　~/book/ch04/stream.py

```
#!/usr/bin/env python
from sys import stdin, stdout
while True:
    line = stdin.readline()
```

```
    if not line:
        break
    stdout.write("%d\n" % int(line)**2)
    stdout.flush()
```

示例 4-8　~/book/ch04/stream.R

```
#!/usr/bin/env Rscript
f <- file("stdin")
open(f)
while(length(line <- readLines(f, n = 1)) > 0) {
        write(as.integer(line)^2, stdout())
}
close(f)
```

4.4　延伸阅读

- Docopt. (2014). Command-Line Interface Description Language. Retrieved from http://docopt.org.
- Robbins, A., & Beebe, N. H. F. (2005). *Classic Shell Scripting*. O'Reilly Media.
- Peek, J., Powers, S., O'Reilly, T., & Loukides, M. (2002). *Unix Power Tools* (3rd Ed.). O'Reilly Media.
- Perkins, J. (2010). *Python Text Processing with NLTK 2.0 Cookbook*. Packt Publishing.
- McKinney, W. (2012). *Python for Data Analysis*. O'Reilly Media.
- Rossant, C. (2013). *Learning IPython for Interactive Computing and Data Visualization*. Packt Publishing.
- Wirzenius, L. (2013). Writing Manual Pages. Retrieved from http://liw.fi/manpages/.
- Raymond, E. S. (2014). Basics of the Unix Philosophy. Retrieved from http://www.faqs.org/docs/artu/ch01s06.html.

第 5 章

数据清洗

第 2 章中，我们学习了数据科学的 OSEMN 模型的第一个步骤，即怎样从多种多样的数据源获取数据。在这些数据中，缺失值、不一致、错误、怪异字符或者冗余列等问题屡见不鲜。有时我们只需要数据的某个部分，有时则需要其他格式的数据。这些情况下，必须清洗数据，然后才能进行第三个步骤：探索数据。

在第 3 章中获取的数据可能有各种各样的格式。最常见的是纯文本、CSV、JSON 和 HTML/XML。多数命令行工具只能处理一种格式，因此能够将数据从一种格式转换为另一种格式是值得的。

CSV 是本章使用的主要格式，但实际上并不是最易用的格式。许多 CSV 数据集受到损坏或者互相不能兼容，因为它不像 XML 和 JSON 那样有标准的语法。

一旦数据是我们想要的格式，就可以对其应用常见的清洗操作，包括数据的过滤、置换和合并。命令行特别适合于这类操作，因为有许多强大的命令行工具，它们都针对海量数据的处理做了优化。本章将讨论 `cut`（Ihnat，MacKenzie & Meyering，2012）和 `sed`（Fenlason，Lord，Pizzini & Bonzini，2012）等经典工具，以及 `jq`（Dolan，2014）和 `csvgrep`（Groskopf，2014）等新工具。

本章讨论的数据清洗任务不仅适用于输入数据。有时还需要对某些命令行工具的输出数据进行格式转换。例如，要将 `uniq -c` 的输出转换为 CSV 格式，可以使用 `awk`（Brennan，1994）和 `header` 命令：

```
$ echo 'foo\nbar\nfoo' | sort | uniq -c | sort -nr
      2 foo
      1 bar
$ echo 'foo\nbar\nfoo' | sort | uniq -c | sort -nr |
> awk '{print $2","$1}' | header -a value,count
value,count
foo,2
bar,1
```

如果除了这些命令行工具（的组合）所提供的功能，你的数据还需要额外的功能，那就可以使用 `csvsql`。这个命令行工具允许你直接对 CSV 文件执行 SQL 查询。如果读完本章你还需要更多的灵活性，记住，你可以自由选择使用 R、Python 或你喜欢的任何程序语言。

我们将在适当授权的基础上介绍命令行工具。你会注意到有时我们可以用同一个命令行工具来执行多个操作，反之亦然，多个命令行工具也可以执行同一个操作。本章的结构更像烹饪书籍，其重点在于问题或者说菜谱，而不是命令行工具。

5.1 概述

本章我们将学习：

- 将数据从一种格式转换为另一种格式
- 对 CSV 应用 SQL 查询
- 行过滤
- 值抽取和替换
- 列分割、合并和抽取

5.2 纯文本的常见清洗操作

本节我们讲述纯文本的常见清洗操作。形式上，纯文本指的是人工可读的字符串，根据需要可以加入一些特定类型的控制字符（例如 tab 符或换行符；更多信息参见 http://www.linfo.org/plain_text.html。这样的例子包括电子书、电子邮件、日志文件和源代码。

为本书的目的起见，我们假设纯文本中包含数据，并且这些数据没有（像 CSV 格式那样的）清晰的表格结构或（像 JSON 和 HTML/XML 格式那样的）嵌套结构。本章后面的部分将讨论这些格式。尽管这些操作也可以应用于 CSV、JSON 和 HTML/XML 格式，但你要记住命令行工具将这些数据看作纯文本。

5.2.1 行过滤

第一个清洗操作是行过滤。这意味要评估输入数据中的每一行来决定是否可以转入输出。

1. 根据位置过滤

最直接的行过滤方法是根据行的位置进行过滤。这在你想要查看文件的（比如）前 10 行，或者从另一个命令行工具的输出中抽取特定列时可能有用。为了解释怎样根据位置进行行过滤，我们创建一个包含 10 行的伪文件：

```
$ cd ~/book/ch05/data
$ seq -f "Line %g" 10 | tee lines
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
```

可以用 head、sed 或 awk 打印前三行：

```
$ < lines head -n 3
$ < lines sed -n '1,3p'
$ < lines awk 'NR<=3'
Line 1
Line 2
Line 3
```

与之类似，可以用 tail（Rubin，MacKenzie，Taylor & Meyering，2012）打印最后三行：

```
$ < lines tail -n 3
Line 8
Line 9
Line 10
```

也可以用 sed 和 awk，但是 tail 要快得多。将前三行删除的命令如下所示：

```
$ < lines tail -n +4
$ < lines sed '1,3d'
$ < lines sed -n '1,3!p'
Line 4
Line 5
Line 6
Line 7
Line 8
Line 9
Line 10
```

请注意 tail 要加 1。删除最后三行可以用 head 命令：

```
$ < lines head -n -3
Line 1
```

```
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
```

可以用 sed 或 awk，也可以组合使用 head 与 tail 来打印（或抽取）特定行（这里是第 4、5、6 行）：

```
$ < lines sed -n '4,6p'
$ < lines awk '(NR>=4)&&(NR<=6)'
$ < lines head -n 6 | tail -n 3
Line 4
Line 5
Line 6
```

用 sed 指定起始行和步长，或者用 awk 的取模运算符来打印奇数行：

```
$ < lines sed -n '1~2p'
$ < lines awk 'NR%2'
Line 1
Line 3
Line 5
Line 7
Line 9
```

用类似的方式打印偶数行：

```
$ < lines sed -n '0~2p'
$ < lines awk '(NR+1)%2'
Line 2
Line 4
Line 6
Line 8
Line 10
```

2. 根据模式过滤

有时可能要根据内容来提取或删除某些行。grep 是行过滤的经典命令行工具，用它可以打印匹配某个模式或正则表达式的所有行。例如，要从 *Alice's Adventures in Wonderland* 中抽取各章标题：

```
$ grep -i chapter alice.txt
CHAPTER I. Down the Rabbit-Hole
CHAPTER II. The Pool of Tears
CHAPTER III. A Caucus-Race and a Long Tale
CHAPTER IV. The Rabbit Sends in a Little Bill
CHAPTER V. Advice from a Caterpillar
CHAPTER VI. Pig and Pepper
CHAPTER VII. A Mad Tea-Party
CHAPTER VIII. The Queen's Croquet-Ground
```

```
CHAPTER IX. The Mock Turtle's Story
CHAPTER X. The Lobster Quadrille
CHAPTER XI. Who Stole the Tarts?
CHAPTER XII. Alice's Evidence
```

这里，-i 的意思是不区分大小写。也可以指定正则表达式。例如，如果只想打印以“The”开头的标题：

```
$ grep -E '^CHAPTER (.*)\. The' alice.txt
CHAPTER II. The Pool of Tears
CHAPTER IV. The Rabbit Sends in a Little Bill
CHAPTER VIII. The Queen's Croquet-Ground
CHAPTER IX. The Mock Turtle's Story
CHAPTER X. The Lobster Quadrille
```

注意，必须指定 -E 选项，这样才能启用正则表达式，否则 grep 会将模式解释为字面字符串。

3. 随机过滤

当你处于构想数据管道的过程中，数据量很大时，调试管道可能会很麻烦。这种情况下，从数据中采样可能有用。命令行工具 sample（Janssens，2014）的主要用途是，通过只将特定比例的输入数据逐行输出，来得到数据的一个子集：

```
$ seq 1000 | sample -r 1% | jq -c '{line: .}'
{"line":53}
{"line":119}
{"line":141}
{"line":228}
{"line":464}
{"line":476}
{"line":523}
{"line":657}
{"line":675}
{"line":865}
{"line":948}
```

这里，每行输入有 1% 的几率输入 jq。也可以将这个比例指定为分数（1/100）或概率（0.01）形式。

sample 还有其他两个用途，当处于调试模式时会派上用场。首先，可以向输出加入延迟。这在输入是持续不断的数据流（例如 Twitter 数据流）以及数据到来得太快而无法看清发生了什么时用得着。其次，可以给 sample 加上定时器。这样，就不必手动终止运行中的进程。给前述命令的每个输出行之间加上 1 秒钟的延迟，并限定运行 5 秒钟：

```
$ seq 10000 | sample -r 1% -d 1000 -s 5 | jq -c '{line: .}'
```

为了避免不必要的计算，应将 sample 尽可能早地放入管道（这个建议对任何缩减数据的命令都适用，如 head 和 tail）。一旦调试结束，可以直接将它移出管道。

5.2.2　值提取

为了从前面的例子中提取实际的各章标题，我们可以采用一种简单的方法：将 grep 的输出通过管道连接到 cut。

```
$ grep -i chapter alice.txt | cut -d' ' -f3-
Down the Rabbit-Hole
The Pool of Tears
A Caucus-Race and a Long Tale
The Rabbit Sends in a Little Bill
Advice from a Caterpillar
Pig and Pepper
A Mad Tea-Party
The Queen's Croquet-Ground
The Mock Turtle's Story
The Lobster Quadrille
Who Stole the Tarts?
Alice's Evidence
```

这里，传递给 cut 的每一行都以空格间隔分成多个字段，然后将第三个字段直至末尾字段输出。各输入行的字段数量可能各不相同。用 sed 也可以完成同样的任务，但是要复杂得多：

```
$ sed -rn 's/^CHAPTER ([IVXLCDM]{1,})\. (.*)$/\2/p' alice.txt > /dev/null
```

（由于输出结果与前面相同，我们通过重定向到 /dev/null 将输出省略掉。）这种方法使用了正则表达式和反向引用。这里，sed 也接管了 grep 的工作。这种复杂方法只有在没有更简单的方法时才是可取的。例如，如果“chapter”是文本正文的一部分，而并不仅用来表明新一章的开始。当然，针对这一问题有许多不同复杂度的方法，这里只是为了阐明一种极端严格的方法。实践中的挑战是，在复杂性和灵活性之间找到一个好的平衡点。

值得注意的是，cut 也可以根据字符位置来分割。这在你想要提取（或删除）每个输入行的相同字符集合时会有用。

```
$ grep -i chapter alice.txt | cut -c 9-
I. Down the Rabbit-Hole
II. The Pool of Tears
III. A Caucus-Race and a Long Tale
IV. The Rabbit Sends in a Little Bill
V. Advice from a Caterpillar
VI. Pig and Pepper
VII. A Mad Tea-Party
VIII. The Queen's Croquet-Ground
IX. The Mock Turtle's Story
X. The Lobster Quadrille
XI. Who Stole the Tarts?
XII. Alice's Evidence
```

grep 有一个非常好的特性，能够将每个匹配结果输出到单独一行：

```
$ < alice.txt grep -oE '\w{2,}' | head
Project
Gutenberg
Alice
Adventures
in
Wonderland
by
Lewis
Carroll
This
```

但是如果要创建以“a”开头并以“e”结尾的所有单词的数据集，应该怎样做？当然还应该用管道：

```
$ < alice.txt tr '[:upper:]' '[:lower:]' | grep -oE '\w{2,}' |
> grep -E '^a.*e$' | sort | uniq -c | sort -nr |
> awk '{print $2","$1}' | header -a word,count | head | csvlook
|-------------+--------|
| word        | count  |
|-------------+--------|
| alice       | 403    |
| are         | 73     |
| archive     | 13     |
| agree       | 11     |
| anyone      | 5      |
| alone       | 5      |
| age         | 4      |
| applicable  | 3      |
| anywhere    | 3      |
| alive       | 3      |
|-------------+--------|
```

5.2.3 值替换和删除

可以使用命令行工具 `tr`（代表“翻译”，即 translate）来替换单个字符。例如，空格可以替换为下划线：

```
$ echo 'hello world!' | tr ' ' '_'
hello_world!
```

如果需要替换多个字符，可以这样组合：

```
$ echo 'hello world!' | tr ' !' '_?'
hello_world?
```

也可以通过指定 `-d` 选项用 `tr` 删除单个字符：

```
$ echo 'hello world!' | tr -d -c '[a-z]'
helloworld
```

这里，我们实际上使用了两个特性。首先，指定了一组字符（所有的小写字母）。其次，指定 `-c` 选项说明使用补集。换句话说，这个命令只保留小写字母。我们甚至可以用 `tr` 将文本变为大写：

```
$ echo 'hello world!' | tr '[a-z]' '[A-Z]'
HELLO WORLD!
$ echo 'hello world!' | tr '[:lower:]' '[:upper:]'
HELLO WORLD!
```

后面的命令更好，因为它还能处理非 ASCII 字符。如果需要处理多个字符，你可能会发现 `sed` 很好用。我们已经看过用 `sed` 从 *Alice in Wonderland* 中抽取各章标题的例子。实际上在 `sed` 中抽取、删除和替换都是同样的操作。你只需要指定不同的正则表达式。例如，要修改一个单词，删除重复的空格，并删除开头的空格：

```
$ echo ' hello     world!' | sed -re 's/hello/bye/;s/\s+/ /g;s/\s+//'
bye world!
```

标记 `g` 代表“全局”（global），意为同样的部分可以在同一行中应用多次。第二部分删除开头的空格，这里我们不再需要它。注意，第一个部分和最后一个部分的正则表达式本来可以组合为一个正则表达式。

5.3 处理CSV

5.3.1 主体、头部和列

我们用来清洗纯文本的命令行工具，如 `tr` 和 `grep`，并不是始终都适用于 CSV 的。原因在于这些命令行工具没有头部、主体和列的概念。如果我们想要使用 `grep` 来对行进行过滤，但是输出中始终包含头部，应该怎么办？或者如果我们只想用 `tr` 将特定列的值变为大写，而保持其他列不变，应该怎么办？对此有多步的变通方案，但是都非常繁琐。其实有更好的办法。为了对处理 CSV 的普通命令行工具加以利用，我们向你介绍三个命令行工具，它们的名称很贴切：`body`（Janssens，2014）、`header`（Janssens，2014）和 `cols`（Janssens，2014）。

让我们从第一个命令行工具 `body` 开始。使用 `body` 可以将任何命令行工具应用到 CSV 文件的主体中（也就是除了头部之外的所有内容）。例如：

```
$ echo -e "value\n7\n2\n5\n3" | body sort -n
value
2
3
5
7
```

`body` 假设 CSV 文件的头只占一行。为了完整性，下面给出了源代码：

```
#!/usr/bin/env bash
IFS= read -r header        ❶
printf '%s\n' "$header"    ❷
$@                         ❸
```

body 是这样工作的：

❶ 从标准输入中提取一行，将其存储为名为 $header 的变量。
❷ 输出头部。
❸ 对剩下的数据执行传递给 body 的所有命令行参数。

下面是另一个例子。假设要计算下列 CSV 文件的行数：

```
$ seq 5 | header -a count
count
1
2
3
4
5
```

使用 wc -l，可以计算所有行的数量：

```
$ seq 5 | header -a count | wc -l
6
```

如果只想考虑主体中的行（也就是除了头部之外的所有内容），只需要加上 body：

```
$ seq 5 | header -a count | body wc -l
count
5
```

注意，没有使用头部，但输出中仍然有它。

第二个命令行工具 header，顾名思义，让我们可以对 CSV 文件的头部进行操作。完整的源代码如下所示：

```
#!/usr/bin/env bash
get_header () {
        for i in $(seq $NUMROWS); do
                IFS= read -r LINE
                OLDHEADER="${OLDHEADER}${LINE}\n"
        done
}

print_header () {
        echo -ne "$1"
}
```

```
print_body () {
        cat
}

OLDHEADER=
NUMROWS=1

while getopts "dn:ha:r:e:" OPTION
do
        case $OPTION in
                n)
                        NUMROWS=$OPTARG
                        ;;
                a)
                        print_header "$OPTARG\n"
                        print_body
                        exit 1
                        ;;
                d)
                        get_header
                        print_body
                        exit 1
                        ;;
                r)
                        get_header
                        print_header "$OPTARG\n"
                        print_body
                        exit 1
                        ;;
                e)
                        get_header
                        print_header "$(echo -ne $OLDHEADER | eval $OPTARG)\n"
                        print_body
                        exit 1
                        ;;
                h)
                        usage
                        exit 1
                        ;;
        esac
done

get_header
print_header $OLDHEADER
```

如果没有提供任何参数，输出的 CSV 文件的头部就是：

```
$ < tips.csv header
bill,tip,sex,smoker,day,time,size
```

这与 `head -n 1` 相同。如果头部占据不止一行（通常不建议这样做），可以将参数指定为 `-n 2`。也可以给 CSV 文件加上头部：

```
$ seq 5 | header -a count
count
1
2
3
4
5
```

这与 `echo "count" | cat - <(seq 5)` 等价。用 `-d` 选项可以将头部删除。

```
$ < iris.csv header -d | head
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

这与 `tail -n +2` 类似，但更容易记忆一些。通过指定 `-r` 就可以对头部进行替换，如果你看一下前面的源代码就会明白，这个操作基本上首先删除头部，然后加入新的头部。这里，我们将其与 `body` 命令组合使用：

```
$ seq 5 | header -a line | body wc -l | header -r count
count
5
```

最后，可以只对头部应用一个命令，这与 `body` 命令行工具对主体所做的类似：

```
$ seq 5 | header -a line | header -e "tr '[a-z]' '[A-Z]'"
LINE
1
2
3
4
5
```

第三个命令行工具称为 `cols`，它与 `header` 和 `body` 类似，都让你可以只对列的子集应用特定命令。其代码如下所示：

```
#!/usr/bin/env bash
ARG="$1"
shift
COLUMNS="$1"
shift
EXPR="$@"
DIRTMP=$(mktemp -d)
mkfifo $DIRTMP/other_columns
```

```
tee $DIRTMP/other_columns | csvcut $ARG $COLUMNS | ${EXPR} |
paste -d, - <(csvcut ${ARG~~} $COLUMNS $DIRTMP/other_columns)
rm -rf $DIRTMP
```

例如，如果想要将 tips.csv 数据集中的 day 列的值变为大写（其他列和头部都不变），应该使用 cols 与 body 的组合，如下所示：

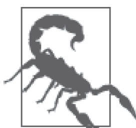
```
$ < tips.csv cols -c day body "tr '[a-z]' '[A-Z]'" | head -n 5 | csvlook
|------+-------+------+--------+--------+--------+-------|
| day  | bill  | tip  | sex    | smoker | time   | size  |
|------+-------+------+--------+--------+--------+-------|
| SUN  | 16.99 | 1.01 | Female | No     | Dinner | 2     |
| SUN  | 10.34 | 1.66 | Male   | No     | Dinner | 3     |
| SUN  | 21.01 | 3.5  | Male   | No     | Dinner | 3     |
| SUN  | 23.68 | 3.31 | Male   | No     | Dinner | 2     |
|------+-------+------+--------+--------+--------+-------|
```

注意，将多个命令行工具和参数作为命令传递给 header -e、body 和 cols 可能导致棘手的引用问题。如果遇到这种问题，最好为此创建单独的命令行工具，然后将其作为命令传递。

总之，尽管使用 CSV 数据专用的命令行工具通常更可取，但如果需要，通过 body、header 和 cols 也可以对 CSV 文件应用经典的命令行工具。

5.3.2 对CSV执行SQL查询

如果本章中提到的命令行工具不够灵活，用命令行工具来清洗数据还有另外一种方法。命令行工具 csvsql（Groskopf，2014）让你可以直接对 CSV 文件执行 SQL 查询。你可能知道，SQL 是定义数据清洗操作的一种非常强大的语言，SQL 与使用单独的命令行工具迥然不同。



如果你的数据最初源于关系数据库，可能的话，可以试着对该数据库执行 SQL 查询，接着将数据抽取为 CSV 格式。正如在第 3 章中讨论的那样，可以使用命令行工具 sql2csv 来做。如果首先将数据从数据库导出到 CSV 文件，然后应用 SQL 查询，那么不只会慢一些，而且从 CSV 数据推测得到的列类型也可能不正确。

在下列清洗任务中，将包含多个涉及 csvsql 的解决方案。基本的命令如下：

```
$ seq 5 | header -a value | csvsql --query "SELECT SUM(value) AS sum FROM stdin"
sum
15
```

如果将标准输入传递给 csvsql，表名就是 stdin。列类型是从数据中自动推测得到的。后面将看到，在 5.5.4 节中，你也可以指定多个 CSV 文件。记住 csvsql 采用 SQLite 方言。尽

管 SQL 通常比其他方案冗长，但它的灵活性要好得多。如果你已经了解怎样用 SQL 处理清洗问题，那么在命令行中使用 SQL 没什么丢脸的！

5.4 处理HTML/XML和JSON

通过第 3 章知道，我们获取的数据可能有各种各样的格式，其中最常见的是纯文本、CSV、JSON 和 HTML/XML。这一节，我们将展示几个能够将数据从一种格式转换为另一种格式的命令行工具。数据转换有两种原因。

首先，数据时常呈现为表格的形式，就像数据库表或电子表格一样，这是因为许多可视化和机器学习算法依赖于表格。CSV 天生就是表格形式的，但 JSON 和 HTML/XML 数据可以有深层嵌套结构。

其次，许多命令行工具，特别是 `cut` 和 `grep` 这样的经典工具，都是在纯文本上进行操作的。这是因为文本被视为命令行工具间的通用接口。而且，其他格式也确实出现得较晚。这些格式每个都可以被看作纯文本，使得我们也可以将这类命令行工具应用到其他格式。

有时我们可以侥幸在结构化数据上成功应用经典工具。例如，通过将 JSON 数据看作纯文本，可以用 `sed` 将属性“gender”改为“sex”：

```
$ sed -e 's/"gender":/"sex":/g' data/users.json | fold | head -n 3
{"results":[{"user":{"sex":"female","name":{"title":"mrs","first":"kaylee","last
":"anderson"},"location":{"street":"1779 washington ave","city":"cupertino","sta
te":"michigan","zip":"13931"},"email":"kaylee.anderson64@example.com","password"
```

像许多其他命令行工具一样，`sed` 并不利用数据的结构。因此，最好使用能对数据结构加以利用的命令行工具（就像我们将用 `jq` 所做的那样），或者首先将数据转换为像 CSV 这样的表格格式，然后应用合适的命令行工具。

接下来，我们将通过一个实际的使用示例来展示从 HTML/XML 和 JSON 向 CSV 的转换。这里将使用的命令行工具是 `curl`、`scrape`（Janssens，2014）、`xml2json`（Parmentier，2014）、`jq`（Dolan，2014）和 `json2csv`（Czebotar，2014）。

维基百科有丰富的信息，其中许多信息是排序成表格的，可以被视为数据集。例如，页面 http://en.wikipedia.org/wiki/List_of_countries_and_territories_by_border/area_ratio 包含了国家和地区的清单，还有它们的边境线长度、面积以及两者的比率。假设我们想分析这个数据集。本节将带你学习所有必要的步骤以及每个步骤对应的命令。

我们感兴趣的数据集是嵌入在 HTML 中的。我们的目的是最终得到能够处理的数据集的表示。第一步就是使用 `curl` 来下载 HTML:

```
$ curl -sL 'http://en.wikipedia.org/wiki/List_of_countries_and_territories_'\
> 'by_border/area_ratio' > wiki.html
```

选项 `-s` 使得 `curl` 进入静默模式，除了实际的 HTML 不输出任何其他信息。HTML 保存到名为 data/wiki.html 的文件中。下面就是该文件前 10 行的内容：

```
$ head -n 10 data/wiki.html | cut -c1-79
<!DOCTYPE html>
<html lang="en" dir="ltr" class="client-nojs">
<head>
<meta charset="UTF-8" /><title>List of countries and territories by border/area
<meta http-equiv="X-UA-Compatible" content="IE=EDGE" /><meta name="generator" c
<link rel="alternate" type="application/x-wiki" title="Edit this page" href="/w
<link rel="edit" title="Edit this page" href="/w/index.php?title=List_of_countr
<link rel="apple-touch-icon" href="//bits.wikimedia.org/apple-touch/wikipedia.p
<link rel="shortcut icon" href="//bits.wikimedia.org/favicon/wikipedia.ico" />
<link rel="search" type="application/opensearchdescription+xml" href="/w/opense
```

这看起来是有顺序的。（注意，每行我们只显示了前 79 个字符，以使输出适合页面宽度。）

使用浏览器的开发工具，能够确定所感兴趣的 HTML 根元素是类 `wikitable` 的 `<table>`。这使得我们可以用 `grep` 来查看我们感兴趣的部分（下面的 `-A` 选项指定我们希望在匹配行之后看到的行数）。

```
$ < wiki.html grep wikitable -A 21
<table class="wikitable sortable">
<tr>
<th>Rank</th>
<th>Country or territory</th>
<th>Total length of land borders (km)</th>
<th>Total surface area (km²)</th>
<th>Border/area ratio (km/km²)</th>
</tr>
<tr>
<td>1</td>
<td>Vatican City</td>
<td>3.2</td>
<td>0.44</td>
<td>7.2727273</td>
</tr>
<tr>
<td>2</td>
<td>Monaco</td>
<td>4.4</td>
<td>2</td>
<td>2.2000000</td>
</tr>
```

下一步是从 HTML 文件中提取必要的元素。这里我们使用 `scrape` 工具：

```
$ < wiki.html scrape -b -e 'table.wikitable > tr:not(:first-child)' \
> > table.html
$ head -n 21 data/table.html
<!DOCTYPE html>
<html>
```

```
<body>
<tr><td>1</td>
<td>Vatican City</td>
<td>3.2</td>
<td>0.44</td>
<td>7.2727273</td>
</tr>
<tr><td>2</td>
<td>Monaco</td>
<td>4.4</td>
<td>2</td>
<td>2.2000000</td>
</tr>
<tr><td>3</td>
<td>San Marino</td>
<td>39</td>
<td>61</td>
<td>0.6393443</td>
</tr>
```

传递给 `-e` 选项（代表“表达式”，即 expression）的值就是所谓的 CSS 选择器（selector）。这个语法经常用来设计网页的样式，但也可以用它从 HTML 中选择特定的元素。这里，我们希望选择属于类 `wikitable` 的表格中所有的 `<tr>` 元素或行（也就是除了第一行的其他行）。这正是我们感兴趣的表格。不想要第一行（通过 `:not(first-child)` 指定）的原因是我们不想要表格的头部。这样得到的结果数据集的每一行代表一个国家或地区。你会看到，现在我们将所查找的 `<tr>` 元素封装在 `<html>` 和 `<body>` 元素中（因为我们指定了 `-b` 选项）。这用来确保下一个工具 `xml2json` 能够对它进行处理。

顾名思义，`xml2json` 将 XML（和 HTML）转换为 JSON 格式。

```
$ < table.html xml2json > table.json
$ < table.json jq '.' | head -n 25
{
  "html": {
    "body": {
      "tr": [
        {
          "td": [
            {
              "$t": "1"
            },
            {
              "$t": "Vatican City"
            },
            {
              "$t": "3.2"
            },
            {
              "$t": "0.44"
            },
            {
```

```
            "$t": "7.2727273"
          }
        ]
      },
      {
        "td": [
```

将 HTML 转换为 JSON 格式的原因在于有一个处理 JSON 的非常强大的工具 `jq`。下列命令提取 JSON 数据的特定部分，将其改造为我们能处理的形式：

```
$ < data/table.json jq -c '.html.body.tr[] | {country: .td[1][],border:'\
> '.td[2][], surface: .td[3][]}' > countries.json
$ head -n 10 data/countries.json
{"surface":"0.44","border":"3.2","country":"Vatican City"}
{"surface":"2","border":"4.4","country":"Monaco"}
{"surface":"61","border":"39","country":"San Marino"}
{"surface":"160","border":"76","country":"Liechtenstein"}
{"surface":"34","border":"10.2","country":"Sint Maarten (Netherlands)"}
{"surface":"468","border":"120.3","country":"Andorra"}
{"surface":"6","border":"1.2","country":"Gibraltar (United Kingdom)"}
{"surface":"54","border":"10.2","country":"Saint Martin (France)"}
{"surface":"2586","border":"359","country":"Luxembourg"}
{"surface":"6220","border":"466","country":"Palestinian territories"
```

现在有点眉目了。JSON 是非常流行的、优点众多的数据格式，但就我们的目的而言，最好还是使用 CSV 格式的数据。工具 `json2csv` 能将数据从 JSON 转换为 CSV 格式：

```
$ < countries.json json2csv -p -k border,surface > countries.csv
$ head -n 11 countries.csv | csvlook
|---------+----------|
| border  | surface  |
|---------+----------|
| 3.2     | 0.44     |
| 4.4     | 2        |
| 39      | 61       |
| 76      | 160      |
| 10.2    | 34       |
| 120.3   | 468      |
| 1.2     | 6        |
| 10.2    | 54       |
| 359     | 2586     |
| 466     | 6220     |
|---------+----------|
```

现在，数据是我们可以处理的形式了。从维基百科页面到 CSV 数据集花费了不少步骤。但是，当将所有这些命令组合为一个命令时，你就会发现它实际上非常简洁，并且具有很强的表达能力。

```
$ curl -sL 'http://en.wikipedia.org/wiki/List_of_countries'\
> '_and_territories_by_border/area_ratio' |
> scrape -be 'table.wikitable > tr:not(:first-child)' |
> xml2json | jq -c '.html.body.tr[] | {country: .td[1][],'\
```

```
> 'border: .td[2][], surface: .td[3][], ratio: .td[4][]}' |
> json2csv -p -k=border,surface | head -n 11 | csvlook
|---------+----------|
| border  | surface  |
|---------+----------|
| 3.2     | 0.44     |
| 4.4     | 2        |
| 39      | 61       |
| 76      | 160      |
| 10.2    | 34       |
| 120.3   | 468      |
| 1.2     | 6        |
| 10.2    | 54       |
| 359     | 2586     |
| 466     | 6220     |
|---------+----------|
```

将 HTML/XML 转换为 JSON 再转换为 CSV 的演示结束了。尽管 jq 可以执行更多操作，尽管有处理 XML 数据的专用工具，但以我们的经验，尽快地将数据转换为 CSV 格式往往比较好。这样，你可以花更多的时间来精通通用的命令行工具而不是非常专用的工具。

5.5 CSV的常见清洗操作

5.5.1 列的提取和重排序

可以用命令行工具 csvcut（Groskopf，2014）对列进行提取和重排序。例如，只保留 Iris 数据集中包含数值的列，并对中间两列重新排序：

```
$ < iris.csv csvcut -c sepal_length,petal_length,sepal_width,petal_width |
> head -n 5 | csvlook
|---------------+--------------+-------------+--------------|
| sepal_length  | petal_length | sepal_width | petal_width  |
|---------------+--------------+-------------+--------------|
| 5.1           | 1.4          | 3.5         | 0.2          |
| 4.9           | 1.4          | 3.0         | 0.2          |
| 4.7           | 1.3          | 3.2         | 0.2          |
| 4.6           | 1.5          | 3.1         | 0.2          |
|---------------+--------------+-------------+--------------|
```

也可以用 -C（代表“互补”，即 complement）指定想要忽略的列：

```
$ < iris.csv csvcut -C species | head -n 5 | csvlook
|---------------+-------------+--------------+--------------|
| sepal_length  | sepal_width | petal_length | petal_width  |
|---------------+-------------+--------------+--------------|
| 5.1           | 3.5         | 1.4          | 0.2          |
| 4.9           | 3.0         | 1.4          | 0.2          |
| 4.7           | 3.2         | 1.3          | 0.2          |
| 4.6           | 3.1         | 1.5          | 0.2          |
|---------------+-------------+--------------+--------------|
```

这里，所包含的列保持相同的顺序。除了列名，也可以指定列的索引号，该索引号从 1 开始。例如，这使得你可以只选择奇数列。（假设你需要它！）

```
$ echo 'a,b,c,d,e,f,g,h,i\n1,2,3,4,5,6,7,8,9' |
> csvcut -c $(seq 1 2 9 | paste -sd,)
a,c,e,g,i
1,3,5,7,9
```

如果你确信任何值中都没有逗号，那还可以使用 cut 来提取列。注意，cut 并不会对列进行排序，如下列命令所示：

```
$ echo 'a,b,c,d,e,f,g,h,i\n1,2,3,4,5,6,7,8,9' | cut -d, -f 5,1,3
a,c,e
1,3,5
```

你可以看到，指定列的顺序无关紧要；使用 cut 会始终保持原来的列顺序。为了完整性起见，让我们也看一下用 SQL 方法对 Iris 数据中的数值列进行提取和重排序的结果：

```
$ < iris.csv csvsql --query "SELECT sepal_length, petal_length, "\
> "sepal_width, petal_width FROM stdin" | head -n 5 | csvlook
|--------------+-------------+------------+-------------|
| sepal_length | petal_length | sepal_width | petal_width |
|--------------+-------------+------------+-------------|
| 5.1          | 1.4         | 3.5        | 0.2         |
| 4.9          | 1.4         | 3.0        | 0.2         |
| 4.7          | 1.3         | 3.2        | 0.2         |
| 4.6          | 1.5         | 3.1        | 0.2         |
|--------------+-------------+------------+-------------|
```

5.5.2　行过滤

与纯文本文件相比，对 CSV 文件进行行过滤的不同之处是，你可能想将过滤只限定于特定列的值。基于位置的过滤本质上是相同的，但你必须考虑到 CSV 文件的第一行通常是头部。记住，如果想要保留头部，你始终可以使用 body 命令行工具：

```
$ seq 5 | sed -n '3,5p'
3
4
5
$ seq 5 | header -a count | body sed -n '3,5p'
count
3
4
5
```

当涉及在特定列中基于特定模式来过滤时，可以使用 csvgrep 和 awk，当然也可以使用 csvsql。例如，要排除所有聚会规模不大于 4 人的账单：

```
$ csvgrep -c size -i -r "[1-4]" tips.csv | csvlook
|--------+------+--------+--------+------+--------+-------|
| bill   | tip  | sex    | smoker | day  | time   | size  |
|--------+------+--------+--------+------+--------+-------|
| 29.8   | 4.2  | Female | No     | Thur | Lunch  | 6     |
| 34.3   | 6.7  | Male   | No     | Thur | Lunch  | 6     |
| 41.19  | 5.0  | Male   | No     | Thur | Lunch  | 5     |
| 27.05  | 5.0  | Female | No     | Thur | Lunch  | 6     |
| 29.85  | 5.14 | Female | No     | Sun  | Dinner | 5     |
| 48.17  | 5.0  | Male   | No     | Sun  | Dinner | 6     |
| 20.69  | 5.0  | Male   | No     | Sun  | Dinner | 5     |
| 30.46  | 2.0  | Male   | Yes    | Sun  | Dinner | 5     |
| 28.15  | 3.0  | Male   | Yes    | Sat  | Dinner | 5     |
|--------+------+--------+--------+------+--------+-------|
```

awk 和 csvsql 也都可以用来进行数值比较。例如，要提取周六或周日中超过 40 美元的所有账单：

```
$ < tips.csv awk -F, '($1 > 40.0) && ($5 ~ /S/)' | csvlook
|--------+------+--------+-----+-----+--------+----|
| 48.27  | 6.73 | Male   | No  | Sat | Dinner | 4  |
|--------+------+--------+-----+-----+--------+----|
| 44.3   | 2.5  | Female | Yes | Sat | Dinner | 3  |
| 48.17  | 5.0  | Male   | No  | Sun | Dinner | 6  |
| 50.81  | 10.0 | Male   | Yes | Sat | Dinner | 3  |
| 45.35  | 3.5  | Male   | Yes | Sun | Dinner | 3  |
| 40.55  | 3.0  | Male   | Yes | Sun | Dinner | 2  |
| 48.33  | 9.0  | Male   | No  | Sat | Dinner | 4  |
|--------+------+--------+-----+-----+--------+----|
```

csvsql 解决方案要繁琐些，但更健壮，因为它使用的是列名而不是列索引号：

```
$ < tips.csv csvsql --query "SELECT * FROM stdin "\
> "WHERE bill > 40 AND day LIKE '%S%'" | csvlook
|--------+------+--------+--------+-----+--------+-------|
| bill   | tip  | sex    | smoker | day | time   | size  |
|--------+------+--------+--------+-----+--------+-------|
| 48.27  | 6.73 | Male   | 0      | Sat | Dinner | 4     |
| 44.3   | 2.5  | Female | 1      | Sat | Dinner | 3     |
| 48.17  | 5.0  | Male   | 0      | Sun | Dinner | 6     |
| 50.81  | 10.0 | Male   | 1      | Sat | Dinner | 3     |
| 45.35  | 3.5  | Male   | 1      | Sun | Dinner | 3     |
| 40.55  | 3.0  | Male   | 1      | Sun | Dinner | 2     |
| 48.33  | 9.0  | Male   | 0      | Sat | Dinner | 4     |
|--------+------+--------+--------+-----+--------+-------|
```

需要注意的是，SQL 查询中 WHERE 子句的灵活性是其他命令行工具难以匹敌的，因为 SQL 可以操作日期和集合，并将子句进行复杂的组合。

5.5.3 列合并

列合并在我们感兴趣的值散布在多个列中时很有用。涉及的列可能是日期（年、月、日可

能分别是独立的列）或名字（姓和名分别是独立的列）。让我们考虑第二种情况。

输入的 CSV 是同一个时代的作曲家的列表。假设我们的任务是将姓和名组合为全名。为此我们将提出四种不同的方法：sed、awk、cols/tr 和 csvsql。让我们看一下输入的 CSV：

```
$ < names.csv csvlook
|-----+-----------+------------+-------|
| id  | last_name | first_name | born  |
|-----+-----------+------------+-------|
| 1   | Williams  | John       | 1932  |
| 2   | Elfman    | Danny      | 1953  |
| 3   | Horner    | James      | 1953  |
| 4   | Shore     | Howard     | 1946  |
| 5   | Zimmer    | Hans       | 1957  |
|-----+-----------+------------+-------|
```

第一种方法 sed 使用分为两部分的表达式。第一部分是替换头部，第二部分是从第二行起开始应用的带有反向引用的正则表达式：

```
$ < names.csv sed -re '1s/.*/id,full_name,born/g;'\
> '2,$s/(.*),(.*),(.*),(.*)/\1,\3 \2,\4/g' | csvlook
|-----+--------------+-------|
| id  | full_name    | born  |
|-----+--------------+-------|
| 1   | John Williams | 1932  |
| 2   | Danny Elfman | 1953  |
| 3   | James Horner | 1953  |
| 4   | Howard Shore | 1946  |
| 5   | Hans Zimmer  | 1957  |
|-----+--------------+-------|
```

awk 方法如下所示：

```
$ < names.csv awk -F, 'BEGIN{OFS=","; print "id,full_name,born"}'\
> '{if(NR > 1) {print $1,$3" "$2,$4}}' | csvlook
|-----+--------------+-------|
| id  | full_name    | born  |
|-----+--------------+-------|
| 1   | John Williams | 1932  |
| 2   | Danny Elfman | 1953  |
| 3   | James Horner | 1953  |
| 4   | Howard Shore | 1946  |
| 5   | Hans Zimmer  | 1957  |
|-----+--------------+-------|
```

cols 方法与 tr 工具组合使用：

```
$ < names.csv cols -c first_name,last_name tr \",\" \" \" |
> header -r full_name,id,born | csvcut -c id,full_name,born | csvlook
|-----+--------------+-------|
| id  | full_name    | born  |
|-----+--------------+-------|
```

```
| 1   | John Williams | 1932  |
| 2   | Danny Elfman  | 1953  |
| 3   | James Horner  | 1953  |
| 4   | Howard Shore  | 1946  |
| 5   | Hans Zimmer   | 1957  |
|-----+--------------+-------|
```

注意，csvsql 利用 SQLite 作为数据库来执行查询，以及 || 代表拼接：

```
$ < names.csv csvsql --query "SELECT id, first_name || ' ' || last_name "\
> "AS full_name, born FROM stdin" | csvlook
|-----+----------------------+-------|
| id  | full_name            | born  |
|-----+----------------------+-------|
| 1   | John Williams        | 1932  |
| 2   | Danny Elfman         | 1953  |
| 3   | James Horner         | 1953  |
| 4   | Howard Shore         | 1946  |
| 5   | Hans Zimmer          | 1957  |
|-----+----------------------+-------|
```

如果 last_name 包含逗号怎么办？清楚起见，让我们看一下初始的输入 CSV：

```
$ cat names-comma.csv
id,last_name,first_name,born
1,Williams,John,1932
2,Elfman,Danny,1953
3,Horner,James,1953
4,Shore,Howard,1946
5,Zimmer,Hans,1957
6,"Beethoven, van",Ludwig,1770
```

看起来前三种方法都无能为力，其失败原因各不相同。只有 csvsql 能够将 first_name 和 full_name 正确地组合起来：

```
$ < names-comma.csv sed -re '1s/.*/id,full_name,born/g;'\
> '2,$s/(.*),(.*),(.*),(.*)/\1,\3 \2,\4/g' | tail -n 1
6,"Beethoven,Ludwig van",1770

$ < names-comma.csv awk -F, 'BEGIN{OFS=","; print "id,full_name,born"}'\
> '{if(NR > 1) {print $1,$3" "$2,$4}}' | tail -n 1
6, van" "Beethoven,Ludwig

$ < names-comma.csv cols -c first_name,last_name tr \",\" \" \" |
> header -r full_name,id,born | csvcut -c id,full_name,born | tail -n 1
6,"Ludwig ""Beethoven van""",1770

$ < names-comma.csv csvsql --query "SELECT id, first_name || ' ' || last_name"\
> " AS full_name, born FROM stdin" | tail -n 1
6,"Ludwig Beethoven, van",1770

$ < names-comma.csv Rio -e 'df$full_name <- paste(df$first_name,df$last_name);'\
> 'df[c("id","full_name","born")]' | tail -n 1
6,"Ludwig Beethoven, van",1770
```

等一下！最后一条命令是什么？是 R 语言吗？事实上，确实是的。这里的 R 代码是通过命令行工具 Rio（Janssens，2014）来评估的。在这个时候我们只能告诉你：这种方法也能成功将两个列合并。后面我们会讨论这个奇妙的工具。

5.5.4 多个CSV文件的合并

1. 垂直拼接

垂直拼接在有些情况下可能是必要的，例如，数据集是每天生成的，或者每个数据集代表一个不同的市场或产品。让我们来模拟后者，将我们钟爱的 Iris 数据集分解成三个 CSV 文件，这样就有了可以再次组合的文件。我们将使用 fieldsplit（Hinds et al.，2010），这是 CRUSH 命令行工具套件的一部分。

```
$ < iris.csv fieldsplit -d, -k -F species -p . -s .csv
```

这里的选项分别是：-d 指定分隔符；-k 指定我们想要保留每个文件的头部；-F 指定某个列，其值决定了允许的输出文件；-p 指定相应的输出文件；-s 指定文件名后缀。由于 Iris 数据集中的 species 列包含三个不同的值，我们最终得到三个 CSV 文件，每个文件有 50 行和一个头部：

```
$ wc -l Iris-*.csv
  51 Iris-setosa.csv
  51 Iris-versicolor.csv
  51 Iris-virginica.csv
 153 total
```

可以用 cat 只向后拼接文件，并用 header -d 命令把除了第一个文件之外的所有文件的头部删除，如下所示：

```
$ cat Iris-setosa.csv <(< Iris-versicolor.csv header -d) \
> <(< Iris-virginica.csv header -d) | sed -n '1p;49,54p' | csvlook
|---------------+-------------+--------------+-------------+------------------|
| sepal_length  | sepal_width | petal_length | petal_width | species          |
|---------------+-------------+--------------+-------------+------------------|
| 4.6           | 3.2         | 1.4          | 0.2         | Iris-setosa      |
| 5.3           | 3.7         | 1.5          | 0.2         | Iris-setosa      |
| 5.0           | 3.3         | 1.4          | 0.2         | Iris-setosa      |
| 7.0           | 3.2         | 4.7          | 1.4         | Iris-versicolor  |
| 6.4           | 3.2         | 4.5          | 1.5         | Iris-versicolor  |
| 6.9           | 3.1         | 4.9          | 1.5         | Iris-versicolor  |
|---------------+-------------+--------------+-------------+------------------|
```

注意，我们使用 sed 只是为了打印第二个文件的头部和前三行的主体信息，来解释成功执行的命令的结果。除了这种方法，还可以使用 csvstack（Groskopf，2014），这种方法更简单，而且不容易出错。

```
$ csvstack Iris-*.csv | sed -n '1p;49,54p' | csvlook
|--------------+------------+-------------+-------------+------------------|
| sepal_length | sepal_width | petal_length | petal_width | species          |
|--------------+------------+-------------+-------------+------------------|
| 4.6          | 3.2        | 1.4         | 0.2         | Iris-setosa      |
| 5.3          | 3.7        | 1.5         | 0.2         | Iris-setosa      |
| 5.0          | 3.3        | 1.4         | 0.2         | Iris-setosa      |
| 7.0          | 3.2        | 4.7         | 1.4         | Iris-versicolor  |
| 6.4          | 3.2        | 4.5         | 1.5         | Iris-versicolor  |
| 6.9          | 3.1        | 4.9         | 1.5         | Iris-versicolor  |
|--------------+------------+-------------+-------------+------------------|
```

如果列 species 列不存在，可以用 csvstack 基于文件名创建一个新列：

```
$ csvstack Iris-*.csv -n species --filenames
```

或者用 -g 指定组名：

```
$ csvstack Iris-*.csv -n class -g a,b,c | csvcut -C species |
> sed -n '1p;49,54p' | csvlook
|--------+-------------+------------+-------------+-------------|
| class  | sepal_length | sepal_width | petal_length | petal_width  |
|--------+-------------+------------+-------------+-------------|
| a      | 4.6         | 3.2        | 1.4         | 0.2         |
| a      | 5.3         | 3.7        | 1.5         | 0.2         |
| a      | 5.0         | 3.3        | 1.4         | 0.2         |
| b      | 7.0         | 3.2        | 4.7         | 1.4         |
| b      | 6.4         | 3.2        | 4.5         | 1.5         |
| b      | 6.9         | 3.1        | 4.9         | 1.5         |
|--------+-------------+------------+-------------+-------------|
```

最前面加入了新列 class。如果你想要改变列顺序，可以如本节前面所讲的那样使用 csvcut。

2. 水平拼接

假设你有三个 CSV 文件，想要把它们拼接在一起。我们在管道的中间使用 tee（Parker，Stallman & MacKenzie，2012）来保存 csvcut 的结果：

```
$ < tips.csv csvcut -c bill,tip | tee bills.csv | head -n 3 | csvlook
|--------+-------|
| bill   | tip   |
|--------+-------|
| 16.99  | 1.01  |
| 10.34  | 1.66  |
|--------+-------|
$ < tips.csv csvcut -c day,time | tee datetime.csv |
> head -n 3 | csvlook
|------+---------|
| day  | time    |
|------+---------|
| Sun  | Dinner  |
| Sun  | Dinner  |
|------+---------|
```

```
$ < tips.csv csvcut -c sex,smoker,size | tee customers.csv |
> head -n 3 | csvlook
|---------+--------+-------|
| sex     | smoker | size  |
|---------+--------+-------|
| Female  | No     | 2     |
| Male    | No     | 3     |
|---------+--------+-------|
```

假设这些行能排成一行，只需使用 `paste`（Ihnat & MacKenzie，2012）将文件拼接起来：

```
$ paste -d, {bills,customers,datetime}.csv | head -n 3 | csvlook
|--------+------+--------+--------+------+-----+---------|
| bill   | tip  | sex    | smoker | size | day | time    |
|--------+------+--------+--------+------+-----+---------|
| 16.99  | 1.01 | Female | No     | 2    | Sun | Dinner  |
| 10.34  | 1.66 | Male   | No     | 3    | Sun | Dinner  |
|--------+------+--------+--------+------+-----+---------|
```

`-d` 选项告诉 `paste` 用逗号作为分隔符。

3. 连接

有时数据不能简单地通过垂直或水平拼接来合并。在某些情况下，特别是关系数据库中，为了减少冗余度，数据分散在多个表或文件中。假设我们想要扩展 Iris 数据集，使其包含关于三类 Iris 鲜花的更多信息，扩展的信息称为 USDA 标识符。这样我们就有了一个关于这些标识符的单独的 CSV 文件：

```
$ csvlook irismeta.csv
|-----------------+----------------------------------------------+----------|
| species         | wikipedia_url                                | usda_id  |
|-----------------+----------------------------------------------+----------|
| Iris-versicolor | http://en.wikipedia.org/wiki/Iris_versicolor | IRVE2    |
| Iris-virginica  | http://en.wikipedia.org/wiki/Iris_virginica  | IRVI     |
| Iris-setosa     |                                              | IRSE     |
|-----------------+----------------------------------------------+----------|
```

这个数据集与 Iris 数据集的共同点是都有 `species` 列。可以用 `csvjoin`（Groskopf，2014）将两个数据集连接起来：

```
$ csvjoin -c species iris.csv irismeta.csv | csvcut -c sepal_length,\
> sepal_width,species,usda_id | sed -n '1p;49,54p' | csvlook
|--------------+-------------+-----------------+----------|
| sepal_length | sepal_width | species         | usda_id  |
|--------------+-------------+-----------------+----------|
| 4.6          | 3.2         | Iris-setosa     | IRSE     |
| 5.3          | 3.7         | Iris-setosa     | IRSE     |
| 5.0          | 3.3         | Iris-setosa     | IRSE     |
| 7.0          | 3.2         | Iris-versicolor | IRVE2    |
| 6.4          | 3.2         | Iris-versicolor | IRVE2    |
| 6.9          | 3.1         | Iris-versicolor | IRVE2    |
|--------------+-------------+-----------------+----------|
```

当然，也可以使用 SQL 方法，即用 `csvsql`，这种方法照例稍显冗长（但可能要灵活得多）：

```
$ csvsql --query 'SELECT i.sepal_length, i.sepal_width, i.species, m.usda_id '\
> 'FROM iris i JOIN irismeta m ON (i.species = m.species)' \
> iris.csv irismeta.csv | sed -n '1p;49,54p' | csvlook
|---------------+------------+-----------------+----------|
| sepal_length  | sepal_width | species         | usda_id  |
|---------------+------------+-----------------+----------|
| 4.6           | 3.2        | Iris-setosa     | IRSE     |
| 5.3           | 3.7        | Iris-setosa     | IRSE     |
| 5.0           | 3.3        | Iris-setosa     | IRSE     |
| 7.0           | 3.2        | Iris-versicolor | IRVE2    |
| 6.4           | 3.2        | Iris-versicolor | IRVE2    |
| 6.9           | 3.1        | Iris-versicolor | IRVE2    |
|---------------+------------+-----------------+----------|
```

5.6 延伸阅读

- Molinaro, A. (2005). *SQL Cookbook*. O’Reilly Media.
- Goyvaerts, J., & Levithan, S. (2012). *Regular Expressions Cookbook* (2nd Ed.). O’Reilly Media.
- Dougherty, D., & Robbins, A. (1997) . *sed & awk* (2nd Ed.). O’Reilly Media.

第 6 章

管理数据工作流

我们希望，到目前为止你已经意识到命令行是做数据科学工作的非常方便的环境。你可能已经注意到，由于使用命令行进行数据科学工作，我们需要：

- 调用许多不同的命令行
- 创建定制的和临时的命令行工具
- 获取和生成许多中间文件

这个过程具有探索性质，导致我们的工作流往往十分混乱，这使得记录以往的工作变得很困难。因此，能够由自己或其他人重现我们曾经执行的步骤十分重要。例如，当继续进行几周前的项目时，我们很有可能已经忘记了对哪些文件，以什么顺序，用什么参数，运行了哪些命令。如果是这样，试想将分析工作转交给你的搭档会多么困难。

可以通过发掘 Bash 的历史数据来找回一些丢失的命令，但这显然不是一个好办法。更好的办法是将命令保存到 Bash 脚本，如 run.sh。这使你和你的搭档至少可以重现分析工作。但是，shell 脚本并非最理想的方法，这是因为：

- 它难以阅读和维护；
- 步骤之间的依赖关系不清晰；
- 所有步骤都一直运行，这样效率不高，有时是不可取的。

这时 Drake（Factual，2014）就可以派上用场。Drake 是由 Factual 创建的命令行工具，它使你可以：

- 以输入和输出的依赖关系的形式将数据工作流步骤形式化

- 使用命令行运行工作流的具体步骤
- 使用内联代码（例如 Pyhon 和 R）
- 从外部数据源存储和获取数据（例如 S3 和 HDFS）

6.1 概述

用 Drake 来管理数据工作流是本章的主题。你将学到：

- 用所谓的 Drakefile 来定义工作流
- 用输入和输出的依赖关系的形式来考虑工作流
- 构建具体目标

6.2 Drake简介

Drake 围绕数据和其中的依赖关系来组织命令的执行。数据处理步骤被形式化为一个单独的文本文件（工作流）。每个步骤通常有一个或多个输入和输出。Drake 自动解析它们之间的依赖关系，并确定需要以什么顺序运行哪些命令。

这意味着，如果有一个需要运行 10 分钟的 SQL 查询，只有当结果消失或查询后来被修改时，才必须要执行该查询。同样，如果你想要运行（或重新运行）一个具体的步骤，Drake 只会考虑运行（或重新运行）这个步骤所依赖的步骤。这可以为你节省大量时间。

拥有形式化的工作流带来的好处是，你可以在几周之后轻易恢复自己的项目，并可以与其他人合作。即使你因为永远无法知道何时会再次运行某些步骤，或者何时在其他的项目中重新使用某些步骤，从而认为这只是个一次性的项目，我们也强烈建议你使用 Drake。

6.3 Drake的安装

Drake 有相当多的依赖关系，这使得它的安装过程相当复杂。在下面的内容中，我们假设你使用的是 Ubuntu 操作系统。



如果你正在使用数据科学工具箱，那么 Drake 已经安装完毕，你大可以跳过本节。

Drake 是用 Clojure 程序语言编写的，这意味着它运行于 Java 虚拟机（JVM）之上。可以找到预构建的 JAR 文件，但由于 Drake 的开发很活跃，因此我们从源代码来构建 Drake。这样，你就需要安装 Leiningen：

```
$ sudo apt-get install openjdk-6-jdk
$ sudo apt-get install leiningen
```

然后，从 Factual 中复制 Drake 库：

```
$ git clone https://github.com/Factual/drake.git
```

并用 Leiningen 构建 JAR 文件：

```
$ cd drake
$ lein uberjar
```

这时创建了 drake.jar 文件。将这个文件复制到你 `PATH` 中的某个目录，例如 ~/.bin：

```
$ mv drake.jar ~/.bin/
```

这时应该已经能够运行 Drake 了：

```
$ cd ~/.bin/
$ java -jar drake.jar
```

这并不见得方便，有两个原因：(1) JVM 的启动时间比较长；(2) 只能从这个目录中运行 Drake。我们建议你安装 Drip，这是 JVM 的启动器，比以 `java` 命令启动得更快。首先，从 Flatland 复制 Drip 库：

```
$ git clone https://github.com/flatland/drip.git
$ cd drip
$ make prefix=~/.bin install
```

然后创建 Bash 脚本，使你可以在任何地方运行 Drake：

```
$ cd ~/.bin
$ cat << 'EOF' > drake
> #!/bin/bash
> drip -cp $(dirname $0)/drake.jar drake.core "$@"
> EOF
$ chmod +x drake
```

要验证是否已经正确安装了 Drake 和 Drip，最好从其他的目录运行下列命令：

```
$ drake --version
Drake Version 0.1.6
```



Drip 能够加速 Java 是因为它在 JVM 运行一次之后保留了 JVM 实例。因此，你会注意到只有第二次之后的运行才会加速。

6.4　获取古腾堡计划中下载最多的电子书

本章后面的内容中，将使用下列任务作为运行示例。目标是将用来解决该任务的命令转变为 Drake 工作流。我们从简单之处着手，然后为了解释 Drake 的各种概念和语法，逐步过渡到更高级的工作流。

古腾堡计划是发起于 1971 年的雄心勃勃的项目，已经将超过 42 000 本书存档和电子化，并全部免费在线提供。在该计划的网站上可以找到下载最多的前 100 本电子书。现在，假设我们对下载最多的前 5 本感兴趣。由于该清单可以以 HTML 形式获取（已经格式化，从而不需要使用 `srape`），获取前 5 本下载最多的电子书就简单直接了：

```
$ cd ~/book/ch06
$ curl -s 'http://www.gutenberg.org/browse/scores/top' | ❶
> grep -E '^<li>' |                                      ❷
> head -n 5 |                                            ❸
> sed -E "s/.*ebooks\/([0-9]+).*/\\1/" > data/top-5      ❹
```

这条命令：

❶ 下载 HTML。

❷ 提取清单项。

❸ 只保留前 5 项。

❹ 将电子书 ID 保存到 data/top-5。

该命令的输出是：

```
$ cat data/top-5
1342
76
11
1661
1952
```

如果想要能在以后重现该命令，最容易做到的就是如第 4 章所讲，将命令放入脚本中。如果再次运行该脚本，也会再次下载这个 HTML 文件。你可能想要控制某些步骤是否要运行，这有三个常见的理由。首先，一个步骤可能需要很长时间。其次，你想要继续处理相同的数据。最后，数据可能来自有速率限制的 API。一个很好的办法是用一个步骤将数据保存到文件，后续步骤对该文件进行操作，这样你就不必再做任何冗余计算或 API 调用了。现在，在我们的例子中第一个理由不成问题，因为 HTML 文件的下载速度足够快。不过，在某些情况下，数据可能来自其他数据源，或者数据量达到千兆字节的规模。

6.5 所有工作流都从单个步骤开始

这一节将把前述的命令转化为 Drake 工作流。一个工作流只是一个文本文件。通常将该文件命名为 Drakefile，因为如果没有在命令行中指定其他文件，Drake 就会使用该文件。只有一个步骤的工作流如示例 6-1 所示。

示例 6-1　只有一个步骤的工作流（Drakefile）

```
data/top-5 <-                                                      ❶
    curl -s 'http://www.gutenberg.org/browse/scores/top' |     ❷
    grep -E '^<li>' |                                          ❸
    head -n 5 |                                                ❹
    sed -E "s/.*ebooks\/([0-9]+).*/\\1/" > data/top-5            ❺
```

让我们把该文件从头到尾看一遍。第一行包含一个左向箭头，这一行是我们的步骤定义。箭头的左边，即 top-5，是这个步骤的名字或输出。该步骤的任何输入都将出现在箭头的右边，但是由于这一步骤没有输入，因此现在是空的。输入和输出的定义使得 Drake 能够识别步骤间的依赖关系，从而能够算出是否需要或者何时执行哪个步骤，才能实现特定的输出。该输出也被称为*目标*（target）。你可以看到，这个步骤的主体实际上是从前的命令，只是加上了缩进。

❶ 箭头（`<-`）表示步骤的名字和其中的依赖关系。后面会对此做更多介绍。

❷ 主体内容是缩进的。

❸ 只选择清单项。

❹ 获取前 5 项。

❺ 提取 ID，将其保存到文件 top-5 中。注意，已经在步骤定义中指定了 top-5，其中的 5 已经使用了三次。后面我们会对此做更多介绍。

这个工作流非常简单。与将命令放到 Bash 脚本中相比，工作流没有任何优势。但不必担心，我们向你保证会有激动人心的事情发生。现在，让我们运行 Drake，看看我们第一个工作流是什么样：

```
$ drake
The following steps will be run, in order:
  1: data/top-5 <- [missing output]
Confirm? [y/n] y
Running 1 steps with concurrence of 1...

--- 0. Running (missing output): data/top-5 <-
--- 0: data/top-5 <- -> done in 0.35s
Done (1 steps run).
```



如果要让 Drake 重新开始，在不同步骤之间删除文件 drake.log、隐藏目录 .drake 以及任何输出文件即可。

如果不指定工作流文件，Drake 就会使用 ./Drakefile。Drake 首先确定需要运行哪些步骤。在我们的例子中，仅有的一个步骤会运行，因为它缺少输出。这意味着没有名为 data/top-5 的文件。Drake 在执行这些步骤之前会发出请求确认的信息。按下回车键，此后很快就会看到 Drake 运行完毕。Drake 没有对我们的步骤报错。现在通过查看输出文件 data/top-5 来验证是否有了前 5 本电子书：

```
$ cat data/top-5
1342
76
11
1661
1952
```

现在我们已经有了输出文件。再次运行 Drake：

```
$ drake
The following steps will be run, in order:
  1: data/top-5 <- [no-input step]
Confirm? [y/n] n
Aborted.
```

可以看到，Drake 想要再次执行这个步骤！但是，现在它提到了一个不同的原因，也就是没有输入步骤（`[no-input-step]`）。它的默认行为是通过查看输入的时间戳来检查输入是否被改变。不过，由于我们没有指定任何输入，Drake 并不知道这个步骤是否应该再次运行。我们可以关闭这个检查时间戳的默认行为，如示例 6-2 所示。

示例 6-2　Drake 工作流的时间检查（01.drake）

```
data/top-5 <- [-timecheck]
    curl -s 'http://www.gutenberg.org/browse/scores/top' |
    grep -E '^<li>' |
    head -n 5 |
    sed -E "s/.*ebooks\/([0-9]+)\">([^<]+)<.*/\\1,\\2/" > data/top-5
```

方括号表明这是该步骤的一个选项。`timecheck` 前面的减号（`-`）意为我们希望将时间戳检查关闭。现在，该步骤只有当缺少输出时才会运行。

让我们使用不同的文件名，以保存旧版本的工作流。可以用 `-w` 来指定另外的工作流名（而不是 Drakefile）。现在再次运行 Drake：

```
$ mv Drakefile 01.drake
$ drake -w 01.drake
Nothing to do.
```

我们最初的工作流已经帮助我们节省了时间，因为 Drake 检测到该步骤不用再次运行。不过，还可以做得比这好得多。这个工作流有三个缺点，我们将在下一节阐述。

6.6　具体情况具体对待

我们的工作流只包含一个步骤，这意味着与拥有简单的 Bash 脚本类似，始终都是全部运行。因此我们要做的第一件事就是将单个步骤分解为两个，其中第一个步骤下载 HTML，第二个步骤处理该 HTML，后者明显依赖于前者。我们可以在工作流中定义这种依赖关系。

你可能已经注意到我们指定了三次数字 5。如果你曾想要获取古腾堡计划的前 10 本电子书，那就得修改工作流的三个地方。这十分低效，需要得到解决。幸好 Drake 支持变量。

数据与脚本放置在相同的地方，这在我们的工作流中可能不是那么明显。最好将数据单独放置，使其与生成该数据的任何代码隔离开来。这不仅可以使项目更干净，还可以方便我们删除生成的数据文件，并且易于说明我们不希望数据文件包含于如 `git`（Torvalds & Hamano，2014）这样的任何版本控制系统中。让我们看一下示例 6-3 中改进的工作流。

示例 6-3　具有依赖关系的 Drake 工作流（02.drake）

```
NUM:=5                                                            ❶
BASE=data/                                                        ❷

top.html <- [-timecheck]
    curl -s 'http://www.gutenberg.org/browse/scores/top' > $OUTPUT  ❸

top-$[NUM] <- top.html                                            ❹
    < $INPUT grep -E '^<li>' |
    head -n $[NUM] |
    sed -E "s/.*ebooks\/([0-9]+)\">([^<]+)<.*/\\1,\\2/" > $OUTPUT
```

❶ 可以在 Drake 中指定变量，最好是在文件的开头，按照变量名、等于号、值的顺序进行变量指定。变量名不必都是大写，但大写确实会让变量更显眼一些。你可以看到，对变量 `NUM` 使用 `:=` 而不是 `=`。这意味着一旦变量 `NUM` 被设置，就不能重写。这使我们可以在运行 Drake 之前通过命令行指定 `NUM` 的值。

❷ 变量 `BASE` 是特殊变量。Drake 会将工作流中指定的所有文件都看作位于这个基本目录中。

❸ 现在我们有两个步骤。第一个步骤跟以前一样有相同的输入，但现在输出是另外的文件，名为 top.html。输出还是定义为步骤 2 的输入。这就是 Drake 知道第二个步骤依赖于第一个步骤的原因所在。

❹ 我们还使用了另外两个特殊变量：`INPUT` 和 `OUTPUT`。这两个特殊变量的值分别设置为对该步骤输入和输出的定义值。这样我们就不必对特定步骤的输入和输出指定两次。而且，它使我们易于重用未来工作流的某些步骤。

现在使用 Drake 来执行新的工作流：

```
$ drake -w 02.drake
The following steps will be run, in order:
  1: data/top.html <-  [missing output]
  2: data/top-5 <- data/top.html [projected timestamped]
```

```
Confirm? [y/n] y
Running 2 steps with concurrence of 1...

--- 0. Running (missing output): data/top.html <-
--- 0: data/top.html <- -> done in 0.89s

--- 1. Running (missing output): data/top-5 <- data/top.html
--- 1: data/top-5 <- data/top.html -> done in 0.02s
Done (2 steps run).
```

假设我们现在想要前 10 本电子书，而不是前 5 本。可以在命令行中设置 `NUM` 变量，并运行 Drake（示例 6-4）。

示例 6-4　NUM=10 时的 Drake 工作流（02.drake）

```
$ NUM=10 drake -w 02.drake
The following steps will be run, in order:
  1: data/top-10 <- data/top.html [missing output]
Confirm? [y/n] y
Running 1 steps with concurrence of 1...

--- 1. Running (missing output): data/top-10 <- data/top.html
--- 1: data/top-10 <- data/top.html -> done in 0.02s
Done (1 steps run).
```

正如你所见，Drake 现在只需要执行第二个步骤，因为第一个步骤的输出已经得到了满足。另外，下载 HTML 文件也不是什么大不了的事情了，但你能想象如果处理的是 10 GB 的数据会有什么后果吗？

6.7　重新构建具体目标

古腾堡计划的前 100 本电子书清单每天都有变化。我们已经看到，如果再次运行 Drake 工作流，包含这个清单的 HTML 不会再次被下载。幸好 Drake 允许我们重新运行特定步骤，这样就可以更新该 HTML 文件：

```
$ drake -w 02.drake '=top.html'
```

相比使用输出文件名来指定想要重新执行哪个步骤，还有一个更方便的方法。可以给步骤的输入和输出都加上所谓的标签（tag）。标签以 `%` 开头。选择简短并且具有描述性的标签名是个好主意，这样易于在命令行中指定标签。让我们给第一个步骤加上标签 `%html`，给第二个步骤加上标签 `%filter`，如示例 6-5 所示。

示例 6-5　带有标签的 Drake 工作流（03.drake）

```
NUM:=5
BASE=data/

top.html, %html <- [-timecheck]
```

```
    curl -s 'http://www.gutenberg.org/browse/scores/top' > $OUTPUT

top-$[NUM], %filter <- top.html
    < $INPUT grep -E '^<li>' |
    head -n $[NUM] |
    sed -E "s/.*ebooks\/([0-9]+)\">([^<]+)<.*/\\1,\\2/" > $OUTPUT
```

现在可以通过指定 `%html` 标签来重新构建第一个步骤：

```
$ drake -w 03.drake '=%html'
```

6.8 讨论

命令行的一个优点是允许你鼓捣数据。你很容易执行不同的命令以及处理不同的数据文件。这是一个极具交互性和反复多次的过程。一段时间之后，你很容易忘记自己执行了哪些步骤才得到所需要的结果。因此每过一会儿就将这些步骤记录下来很重要。这样，如果过段时间你自己或你的某个搭档接手这个项目，可以通过执行相同的步骤来生成相同的结果。

本章展示了只将所有命令放入一个 Bash 脚本的做法并不理想。我们提出用 Drake 作为命令行工具来管理数据工作流。通过使用一个运行示例，我们展示了怎样定义步骤和步骤间的依赖关系，还讨论了怎样使用变量和标签。

没有比忘记其他一切专心鼓捣数据更有意思的事了。但是请相信我，保留所有事情的记录（借助于 Drake 工作流）是值得的。它不但会让你的生活变得更简单，而且还会使你开始以步骤的形式来思考数据工作流。正如使用自己的数据科学工具箱那样——经过一段时间的不断扩充，能够使你更加高效——Drake 工作流也有助于让你的方案更有条理。定义的步骤越多，继续这样做就越容易，因为经常可以重用某些步骤。希望你能够习惯使用 Drake，它将使你的生活更轻松。

本章只涉及了 Drake 的一点皮毛。Drake 的一些更高级的特性包括：

- 步骤的异步执行
- 对 Python 和 R 内联代码的支持
- 从 HDFS 和 S3 上传和下载数据

6.9 延伸阅读

- Factual. (2014). Drake. Retrieved from https://github.com/Factual/drake.

第 7 章

数据探索

既然已经得到了数据并对它进行了清洗，我们就可以继续进行 OSEMN 模型的第三步了，也就是数据探索。在完成前面那些困难工作之后，（除非你已经拥有了干净的数据！）是时候放松一下了。

数据探索是让你熟悉数据的一个步骤。当你想要从数据中获取价值时，熟悉数据是必不可少的。例如，了解数据有哪些特征，意味着你知道哪些数据值得进一步探索，哪些数据可以用于解决你的问题。

数据探索可以通过三个方面实现。第一个方面是检查数据及其属性。例如，在这里，我们想知道原始数据是什么样子，数据集里有多少数据，以及数据集有哪些特征。

数据探索的第二个方面是计算描述性统计信息。这一点对于了解更多关于单个特征的信息比较有用处。其优点在于，它的输出通常很简明，而且是文字性的，因而可以在命令行中打印出来。

第三个方面是为数据生成可视化图形。在这里，我们深入了解关于多个特征如何交互的信息。我们会讨论一种可以在命令行中打印的数据可视化生成方式。然而，多数的可视化是在图形用户界面上显示的。数据可视化相对于描述性统计信息的一个优点在于，数据可视化更加灵活，可以传递更多信息。

7.1 概述

本章你将学习：

- 检查数据及其属性
- 计算描述性统计信息
- 在命令行之内和之外生成数据可视化图形

7.2 检查数据及其属性

在本节中，我们将会通过实例说明如何检查一个数据集及其属性。由于后续的可视化和建模技术都需要以表格形式存在的数据，因此我们假设数据是 CSV 格式的。如果必要的话，你可以使用第 5 章介绍的技术把数据转换成 CSV 格式。

为了简单起见，我们还假设数据里有一个数据头。在 7.2.1 节中，我们会检查数据是否有数据头。一旦数据就绪，我们就可以继续回答如下问题：

- 数据集里有多少数据和特征？
- 原始数据是什么样的？
- 数据集有哪些特征？
- 其中一些特征是否能看作类别特征或者因子？

7.2.1 确定有无数据头

你可以通过打印数据的前几行，检查你的文件中是否含有数据头：

```
$ head file.csv | csvlook
```

第一行是数据头还是第一个数据，由你决定。当数据集里不包含数据头或者其数据头含有换行符的时候，你最好回到前面的步骤，通过数据清洗来修正（参考第 5 章了解如何清洗数据）。

7.2.2 检查所有数据

如果你想检查原始数据，那么最好不要使用 `cat` 命令行工具，这是因为 `cat` 会在屏幕上一次性打印出所有数据。为了按照你自己的显示速度来查看原始数据，我们推荐使用带有 `-S` 选项的 `less`（Nudelman，2013）。

```
$ less -S file.csv
```

当较长的行无法适应终端显示时，`-S` 选项可以保证它们不会卷起。作为替代，`less` 允许你横向滚动，查看这些行剩下的内容。`less` 的优点在于，它并不会把整个文件都读进内存，这对于浏览大型文件是有好处的。一旦你处于 `less` 状态中，可以通过按下空格键向下滚动一整屏。横向滚动则可以通过按下向左和向右键实现。按 `g` 和 `G` 可以分别跳到文件的开头和末尾。按 `q` 可以退出 `less`。在操作说明页面中可以找到更多的按键组合。

如果你想要让数据集具有漂亮的格式，可以把 csvlook 加入到管道里：

```
$ < file.csv csvlook | less -S
```



不巧的是，csvlook 需要将整个文件都读到内存里来确定列的宽度。所以，当你查看一个非常庞大的文件时，可能只愿意获取一个子集（例如使用 sample），或者需要耐心等待一会儿。

7.2.3 特征名称和数据类型

为了深入了解数据集，将特征名称打印出来并研究一下是很有意义的。毕竟，特征名称可能暗含特征的含义。你可以使用如下 sed 表达式来做这件事：

```
$ cd ~/book/ch07
$ < data/iris.csv sed -e 's/,/\n/g;q'
sepal_length
sepal_width
petal_length
petal_width
species
```

注意，这个基本命令假设文件是以逗号分隔的。提醒一下，如果你想经常使用这个命令，可以在 ~/.bashrc 文件里定义一个函数，叫作 names：

```
names () { sed -e 's/,/\n/g;q'; }
```

你可以这样来使用：

```
$ < data/investments2.csv names
company_permalink
company_name
company_category_list
company_market
company_country_code
company_state_code
company_region
company_city
investor_permalink
investor_name
investor_category_list
investor_market
investor_country_code
investor_state_code
investor_region
investor_city
funding_round_permalink
funding_round_type
funding_round_code
funded_at
```

```
funded_month
funded_quarter
funded_year
raised_amount_usd
```

我们还可以比打印每列名称更进一步。除了这些列的名称，了解每列中所包含数据的类型也是非常有用的。数据类型的例子有字符串、数值或者日期。假设我们有如下简单数据集：

```
$ < data/datatypes.csv csvlook
|-----+--------+-------+----------+-----------------+------------+----------|
|  a  |  b     |  c    |  d       |  e              |  f         | g        |
|-----+--------+-------+----------+-----------------+------------+----------|
|  2  |  0.0   | False | "Yes!"   | 2011-11-11 11:00 | 2012-09-08 | 12:34    |
|  42 | 3.1415 | True  | Oh, good | 2014-09-15      | 12/6/70    | 0:07 PM  |
|  66 |        | False | 2198     |                 |            |          |
|-----+--------+-------+----------+-----------------+------------+----------|
```

我们在第 5 章里已经使用过 csvql 直接对 CSV 数据执行 SQL 请求。倘若我们想要把这条数据插入到实际数据库中，它就会在没有任何命令行参数传递进来的时候生成所需要的 SQL 语句。我们还可以用它的输出来查看有哪些列类型：

```
csvsql data/datatypes.csv
CREATE TABLE datatypes (
        a INTEGER NOT NULL,
        b FLOAT,
        c BOOLEAN NOT NULL,
        d VARCHAR(8) NOT NULL,
        e DATETIME,
        f DATE,
        g TIME,
        CHECK (c IN (0, 1))
);
```

表 7-1 给出了各种 SQL 数据类型含义的概述。如果在一个列中，数据类型后面有 NOT NULL 字符串，那么该列就不会包括缺失值。

表7-1　Python与SQL数据类型

类　型	Python	SQL
字符串	unicode	VARCHAR
布尔型	bool	BOOLEAN
整数	int	INTEGER
实数	float	FLOAT
日期	datetime.date	DATE
时间	datetime.time	TIME
日期和时间	datetime.datetime	DATETIME

7.2.4 唯一标识、连续变量和因子

知道每个特征的数据类型是不够的。了解每个特征所代表的意义也是必不可少的。虽然掌握领域知识很有用处，但我们还可以从数据本身获得一些灵感。

字符串和整数都可以用来表示唯一标识或代表一个类别。对于后者，它可以用于给可视化图形赋予一种颜色。如果一个整数表示 ZIP 代码，那么计算平均值就是没有任何意义的。

要确定一个特征能否被当作一个唯一标识或者类别变量（或者是 R 术语里面的"factor"），你可以统计某个特定列中不同数值的个数：

```
$ cat data/iris.csv | csvcut -c species | body "sort | uniq | wc -l"
species
3
```

或者你可以使用 `csvstat`（Groskopf，2014）来获得每列中不同值的个数。`csvstat` 是 Cvskit 中的一部分。

```
$ csvstat data/investments2.csv --unique
  1. company_permalink: 27342
  2. company_name: 27324
  3. company_category_list: 8759
  4. company_market: 443
  5. company_country_code: 150
  6. company_state_code: 147
  7. company_region: 1079
  8. company_city: 3305
  9. investor_permalink: 11176
 10. investor_name: 11135
 11. investor_category_list: 468
 12. investor_market: 134
 13. investor_country_code: 111
 14. investor_state_code: 80
 15. investor_region: 549
 16. investor_city: 1198
 17. funding_round_permalink: 41790
 18. funding_round_type: 13
 19. funding_round_code: 15
 20. funded_at: 3595
 21. funded_month: 295
 22. funded_quarter: 121
 23. funded_year: 34
 24. raised_amount_usd: 6143
```

如果不同值的数量比数据集中的行数还少，那这个特征可能确实应该被当作一个类别特征（例如 `funding_round_type`）。如果这个数量等于行数，那它可以作为唯一标识（例如 `company_permalink`）。

7.3 计算描述性统计信息

7.3.1 使用csvstat

命令行工具 `csvstat` 给出了很多信息。对于每个特征，它显示了：

- Python 术语中的数据类型（见表 7-1 中的 Python 和 SQL 数据类型比较）。
- 是否含有缺失值（`Null`）。
- 不同值的个数。
- 有关适用特征的各种描述性统计信息（也就是最大值、最小值、和、均值、标准偏差以及中值）。

我们按照如下方式使用 `csvstat`：

```
$ csvstat data/datatypes.csv
  1. a
        <type 'int'>
        Nulls: False
        Values: 2, 66, 42
  2. b
        <type 'float'>
        Nulls: True
        Values: 0.0, 3.1415
  3. c
        <type 'bool'>
        Nulls: False
        Unique values: 2
        5 most frequent values:
                False:  2
                True:   1
  4. d
        <type 'unicode'>
        Nulls: False
        Values: 2198, "Yes!", Oh, good
  5. e
        <type 'datetime.datetime'>
        Nulls: True
        Values: 2011-11-11 11:00:00, 2014-09-15 00:00:00
  6. f
        <type 'datetime.date'>
        Nulls: True
        Values: 2012-09-08, 1970-12-06
  7. g
        <type 'datetime.time'>
        Nulls: True
        Values: 12:34:00, 12:07:00

Row count: 3
```

这里给出了非常冗长的结果。要得到更简明的输出，就要指定如下统计信息中的一个选项：

- `--max`（最大值）
- `--min`（最小值）
- `--sum`（和）
- `--mean`（均值）
- `--median`（中值）
- `--stdev`（标准偏差）
- `--nulls`（列中是否包含空值）
- `--unique`（不同值）
- `--freq`（频繁值）
- `--len`（数值的最大长度）

例如：

```
$ csvstat data/datatypes.csv --null
  1. a: False
  2. b: True
  3. c: False
  4. d: False
  5. e: True
  6. f: True
  7. g: True
```

你可以用 `-c` 选项选取特征的一个子集。它既可以是整数，也可以是列的名称。

```
$ csvstat data/investments2.csv -c 2,13,19,24
  2. company_name
        <type 'unicode'>
         Nulls: True
        Unique values: 27324
        5 most frequent values:
                Aviir: 13
                Galectin Therapeutics: 12
                Rostima:        12
                Facebook:       11
                Lending Club:   11
        Max length: 66
 13. investor_country_code
        <type 'unicode'>
        Nulls: True
        Unique values: 111
         5 most frequent values:
                USA:    20806
                GBR:    2357
                DEU:    946
                CAN:    893
                FRA:    737
        Max length: 15
 19. funding_round_code
        <type 'unicode'>
        Nulls: True
```

```
        Unique values: 15
        5 most frequent values:
                a:      7529
                b:      4776
                c:      2452
                d:      1042
                e:      384
        Max length: 10
 24. raised_amount_usd
        <type 'int'>
        Nulls: True
        Min: 0
        Max: 3200000000
        Sum: 359891203117
        Mean: 10370010.1748
        Median: 3250000
        Standard Deviation: 38513119.1802
        Unique values: 6143
        5 most frequent values:
                10000000:      1159
                1000000:       1074
                5000000:       1066
                2000000:       875
                3000000:       820

Row count: 41799
```

注意，csvstat 像 csvsql 一样使用启发式方法来确定数据类型，因此它并不是总能得到正确的结果。我们鼓励你一直都做人工检查，就如前面讨论的那样。此外，数据类型可以是字符串或者整数，但是并没有说明应当将它当作哪种类型。

作为一个不错的副产品，csvstat 会在最后输出数据点的个数。换行符和数值中的逗号都会被正确处理。我们可以用 tail 来查看那些相关的输出行：

```
$ csvstat data/iris.csv | tail -n 1
Row count: 150
```

如果你只想查看数据点的实际个数，则可以使用如下 sed 表达式来获得这个值：

```
$ csvstat data/iris.csv | sed -rne '${s/^([^:]+): ([0-9]+)$/\2/;p}'
150
```

7.3.2　在命令行中通过Rio使用R

在本节中，我们将会介绍一个命令行工具，叫作 Rio。它是统计编程环境 R 中的一款小巧精致的程序包。在说明 Rio 是什么以及它为什么会出现之前，让我们先说一些有关 R 本身的内容。

R 是一个功能非常强大的统计软件包，用于分析数据和生成可视化图形。它是一种解释型编程语言，拥有大量的程序包，并提供了类似于命令行的 REPL，让你试验数据。不幸的

是，R 与命令行是分离的。一旦你开始使用 R，就处于一个独立的环境中。R 并不能真正与命令行很好地协作，这是因为你无法将任何数据通过管道输入到 R 中，它也并不支持你所给出的任何单行命令。

例如，假设你有一个 CSV 文件，叫作 data/tips.csv，你想要计算小费的比例，并保存结果。要用 R 完成这件事，你需要先启动 R：

```
$ R
```

然后运行如下命令：

```
> tips <- read.csv('data/tips.csv', header = T, sep = ',', stringsasfactors = F)
> tips.percent <- tips$tip / tips$bill * 100
> cat(tips.percent, sep = '\n', file = 'data/percent.csv')
> q("no")
```

之后，你可以在命令行中继续使用保存下来的 data/percent.csv 文件。注意，这里只有一条命令和你想要实现的功能相关。其他命令都是必要的模板代码。敲入这些模板代码来实现一些简单的功能是比较麻烦的，而且这样打断了你的工作流。有时候，你只想每次对数据做一两件事情。如果能够利用 R 的强大功能，并且可以在命令行中使用它，这不是一件非常棒的事情吗？

`Rio` 在这里登场了。`Rio` 的名字代表的是“R input/output”，这是由于它能让你把 R 当作一个过滤器在命令行中使用。你只需要把 CSV 数据通过管道输入到 `Rio` 中，并给出你想在数据上运行的 R 命令即可。让我们再做一次之前那个任务，但这次使用 `Rio`：

```
$ < data/tips.csv Rio -e 'df$tip / df$bill * 100' | head
5.944673
16.05416
16.65873
13.97804
14.68076
18.62396
22.80502
11.60714
13.03191
21.85386
```

`Rio` 能够运行通过分号分隔的多个 R 命令。所以，如果你想要在输入数据中增加一个叫作 percent 的列，可以做如下事情：

```
$ < data/tips.csv Rio -e 'df$percent <- df$tip / df$bill * 100; df' | head
bill,tip,sex,smoker,day,time,size,percent
16.99,1.01,Female,No,Sun,Dinner,2,5.94467333725721
10.34,1.66,Male,No,Sun,Dinner,3,16.0541586073501
21.01,3.5,Male,No,Sun,Dinner,3,16.6587339362208
23.68,3.31,Male,No,Sun,Dinner,2,13.9780405405405
24.59,3.61,Female,No,Sun,Dinner,4,14.6807645384303
```

```
25.29,4.71,Male,No,Sun,Dinner,4,18.6239620403321
8.77,2,Male,No,Sun,Dinner,2,22.8050171037628
26.88,3.12,Male,No,Sun,Dinner,4,11.6071428571429
15.04,1.96,Male,No,Sun,Dinner,2,13.031914893617
```

使用这些单行命令是可行的，这是由于 Rio 可以处理所有模板代码。能够使用命令行，并且在单行命令中具备 R 的功能，这是相当有吸引力的，特别是当你想要一直使用命令行工作的时候。Rio 假设输入数据是 CSV 格式的，并且包含数据头。（通过指定 -n 选项，Rio 不会把第一行当作数据头，然后生成默认的列名称。）在后台，Rio 会把从管道输入的数据写入一个临时的 CSV 文件中，并生成一个脚本：

- 引入所需的程序包
- 把 CSV 文件读到 data.frame 中
- 如果需要的话，生成一个 ggplot2 对象（更多内容见下一节）
- 运行指定的命令
- 将最后一个命令的结果打印到标准输出中

现在，如果你想用 R 对数据集做一两个操作，可以使用单行命令，并一直使用命令行工作。你对 R 所了解的所有知识，现在都可以在命令行中使用。通过 Rio，你甚至可以生成复杂的可视化图形，如同在本章后面所看到的那样。

Rio 并不是必须被当作过滤器使用，这意味着输出本身并不是一定要是 CSV 格式的。你可以计算各种描述性统计信息：

```
$ < data/iris.csv Rio -e 'mean(df$sepal_length)'
5.843333
$ < data/iris.csv Rio -e 'sd(df$sepal_length)'
0.8280661
$ < data/iris.csv Rio -e 'sum(df$sepal_length)'
876.5
```

如果想要计算 5 种汇总统计，我们可以：

```
$ < data/iris.csv Rio -e 'summary(df$sepal_length)'
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900
```

你还可以计算偏度（分布的对称性）和峰度（分布的峰值），但需要安装 moments 包：

```
$ < data/iris.csv Rio -e 'skewness(df$sepal_length)'
$ < data/iris.csv Rio -e 'kurtosis(df$petal_width)'
```

两个特征之间的相关性：

```
$ < dat/iris.csv Rio -e 'cor(df$bill, df$tip)'
0.6757341
```

甚至是相关矩阵：

```
$ < data/tips.csv csvcut -c bill,tip | Rio -f cor | csvlook
|--------------------+--------------------|
|  bill              |  tip               |
|--------------------+--------------------|
|  1                 |  0.675734109211365 |
|  0.675734109211365 |  1                 |
|--------------------+--------------------|="">
```

注意，我们可以通过 `-f` 选项指定用于 `data.frame` 的函数 `df`。在这种情况下，它与 `-e cor(df)` 的功能相同。

你甚至可以用 `Rio` 生成一个茎状图形（Tukey，1977）：

```
$ < data/iris.csv Rio -e 'stem(df$sepal_length)'

  The decimal point is 1 digit(s) to the left of the |

  42 | 0
  44 | 0000
  46 | 000000
  48 | 00000000000
  50 | 0000000000000000000
  52 | 00000
  54 | 0000000000000
  56 | 00000000000000
  58 | 0000000000
  60 | 000000000000
  62 | 0000000000000
  64 | 000000000000
  66 | 0000000000
  68 | 0000000
  70 | 00
  72 | 0000
  74 | 0
  76 | 00000
  78 | 0
```

7.4 生成可视化图形

在本节中，我们将会讨论如何用命令行生成可视化图形。我们将会看看两个不同的软件包：Gunplot 和 `ggplot2`。首先，我们会介绍一下这两个软件包，然后用实例说明如何用它们生成不同类型的可视化图形。

7.4.1 介绍Gunplot和feedgnuplot

我们将在本章里讨论的第一个生成可视化图形的软件包是 Gunplot，它创建于 1986 年左右。尽管相当古老，但它的可视化能力相当广泛。因此，在一节中尽言其意是不太可能的。目前已经有不少优秀资源可以使用，其中包括 *Gunplot in Action* 这本书，由 Janert 于 2009 年创作。

要说明它的灵活性（以及它的古老符号），见示例 7-1，这是从 Gunplot 网站（http://gunplot.sourceforge.net/demo/histograms.6gnu）复制过来的。

示例 7-1　用 Gunplot 生成一个直方图

```
# set terminal pngcairo  transparent enhanced font "arial,10" fontscale 1.0 size
# set output 'histograms.6.png'
set border 3 front linetype -1 linewidth 1.000
set boxwidth 0.75 absolute
set style fill   solid 1.00 border lt -1
set grid nopolar
set grid noxtics nomxtics ytics nomytics noztics nomztics \
 nox2tics nomx2tics noy2tics nomy2tics nocbtics nomcbtics
set grid layerdefault   linetype 0 linewidth 1.000, linetype 0 linewidth 1.000
set key outside right top vertical Left reverse noenhanced autotitles columnhead
set style histogram columnstacked title offset character 0, 0, 0
set datafile missing '-'
set style data histograms
set xtics border in scale 1,0.5 nomirror norotate offset character 0, 0, 0 auto
set xtics norangelimit
set xtics  ()
set ytics border in scale 0,0 mirror norotate offset character 0, 0, 0 autojust
set ztics border in scale 0,0 nomirror norotate offset character 0, 0, 0 autoju
set cbtics border in scale 0,0 mirror norotate offset character 0, 0, 0 autojus
set rtics axis in scale 0,0 nomirror norotate offset character 0, 0, 0 autojust
set title "Immigration from Northern Europe\n(columstacked histogram)"
set xlabel "Country of Origin"
set ylabel "Immigration by decade"
set yrange [ 0.00000 : * ] noreverse nowriteback
i = 23
plot 'immigration.dat' using 6 ti col, '' using 12 ti col,     '' using 13 ti c
```

注意，上述代码被整理成 80 个字符宽。这段脚本生成的图形见图 7-1。

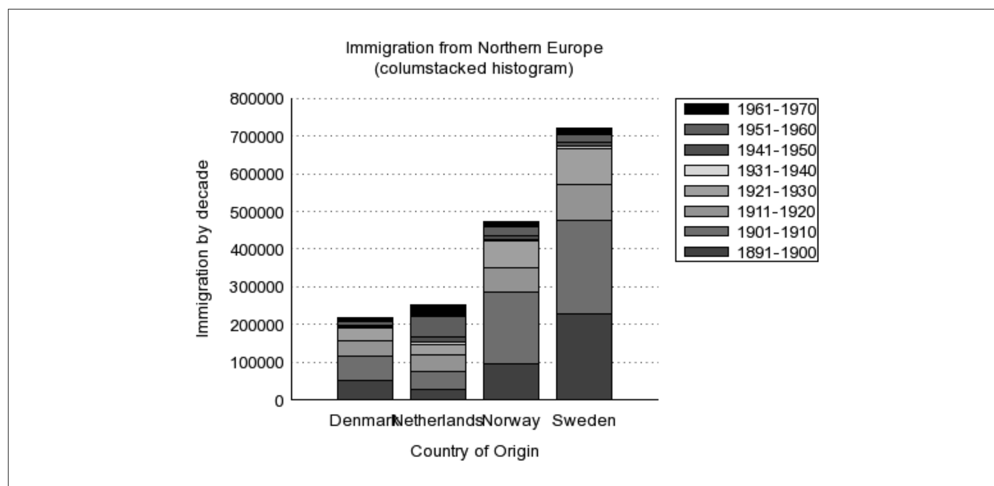


图 7-1：用 Gunplot 绘制的移民量图

Gunplot 和我们使用过的绝大多数命令行工具都不一样。原因有两个。首先，它使用脚本，而不是命令行参数。其次，它的输出都会写到文件里，而不是打印到标准输出中。

Gunplot 的一个很大优势在于，它能够为命令行生成可视化图形，这也是为什么它会长期存在，以及我们把它囊括到本书中的主要原因。这是说，它能把输出打印到终端上，而不需要 GUI。即便如此，我们仍需要创建一个脚本。

幸运的是，有一个命令行工具叫作 `feedgunplot`（Kogan，2014），它可以帮我们为 Gunplot 创建脚本。通过命令行参数，`feedgunplot` 完全可配置。外加上，它能从标准输入中读取数据。介绍完 `ggplot2` 之后，我们会用 `feedgunplot` 生成一些可视化图形。

我们想在这里介绍 `feedgunplot` 的一个优秀特性是，它能让你为流式数据绘图。下面是一个基于随机输入数据的连续更新的绘图片段：

```
$ while true; do echo $RANDOM; done | sample -d 10 | feedgnuplot --stream \
> --terminal 'dumb 80,25' --lines --xlen 10

  30000 ++-----+------------+-------------+-------------+------------+-----++
        |      +            *             +             +            +      |
        |      :            **            :             *******      :      *
  25000 ++.................*.*..........................*.....*............+*
        |      :           *: *           :            *:      *     :     *|
        |      :           *: *           :            *:      *     :     *|
        |      :          * :  *          :           * :       *    :    * |
  20000 ++................*....*......................*.........*.........*++
        |      :          * :   *         :          *  :       *    :    * |
        |      :         *  :   *         :          *  :        *   :   *  |
  15000 ++....**.........*.......*..................*............*.......*.++
        | **** :*        *  :    *        :         *   :         *  :   *  |
        **     :*        *  :     *      ****      *    :         *  :  *   |
  10000 ++.......*......*.........*....**....*.....*..............*.....*..++
        |      :  *     *   :      * **   :   *   *     :          * : *    |
        |      :   *    *   :      **     :    ** *     :          * : *    |
        |      :   *   *    :             :      *      :          * : *    |
   5000 ++..........*..*.........................*..................*.*....++
        |      :     * *    :             :             :           *:*     |
        |      +     **     +             +             +            *      |
      0 ++-----+------*-----+-------------+-------------+------------*-----++
              2350         2352          2354          2356         2358
```

7.4.2 介绍ggplot2

有一个用于生成可视化图形的更现代的软件包叫作 `ggplot2`。它是 R 图形语法的一个实现（Wickham，2009）。

感谢图形语法，通过使用恰当的默认值，`ggplot2` 命令非常短小精悍。在 `Rio` 里使用它，是在命令行中生成可视化图形的一种非常便利的方式。

为了说明它的表达能力，我们借助 Rio 重新生成了之前用 Gnuplot 生成过的直方图。由于 Rio 需要数据集以逗号分隔，而 ggplot2 需要 long 格式的数据，因此我们首先要对数据做一点点清洗和转换：

```
$ < data/immigration.dat sed -re '/^#/d;s/\t/,/g;s/,-,/,0,/g;s/region/'\
> 'Period/' | tee data/immigration.csv | head | cut -c1-80
Period,Austria,Hungary,Belgium,Czechoslovakia,Denmark,France,Germany,Greece,Irel
1891-1900,234081,181288,18167,0,50231,30770,505152,15979,388416,651893,26758,950
1901-1910,668209,808511,41635,0,65285,73379,341498,167519,339065,2045877,48262,1
1911-1920,453649,442693,33746,3426,41983,61897,143945,184201,146181,1109524,4371
1921-1930,32868,30680,15846,102194,32430,49610,412202,51084,211234,455315,26948,
1931-1940,3563,7861,4817,14393,2559,12623,144058,9119,10973,68028,7150,4740,3960
1941-1950,24860,3469,12189,8347,5393,38809,226578,8973,19789,57661,14860,10100,1
1951-1960,67106,36637,18575,918,10984,51121,477765,47608,43362,185491,52277,2293
1961-1970,20621,5401,9192,3273,9201,45237,190796,85969,32966,214111,30606,15484,
```

sed 表达式包含 4 部分，通过分号分隔：

(1) 删除以注释（#）开头的行。
(2) 将 tab 转换为逗号。
(3) 将破折号（缺失值）变为 0。
(4) 将特征名 Region 变为 Period。

我们用 csvcut 只选择那些需要的列，然后使用 Rio 和 melt 函数（它是 R 程序包 reshape2 的一部分）把数据从宽格式变为长格式：

```
$ < data/immigration.csv csvcut -c Period,Denmark,Netherlands,Norway,\
> Sweden | Rio -re 'melt(df, id="Period", variable.name="Country", '\
> 'value.name="Count")' | tee data/immigration-long.csv | head | csvlook
|------------+-------------+--------|
| Period     | Country     | Count  |
|------------+-------------+--------|
| 1891-1900  | Denmark     | 50231  |
| 1901-1910  | Denmark     | 65285  |
| 1911-1920  | Denmark     | 41983  |
| 1921-1930  | Denmark     | 32430  |
| 1931-1940  | Denmark     | 2559   |
| 1941-1950  | Denmark     | 5393   |
| 1951-1960  | Denmark     | 10984  |
| 1961-1970  | Denmark     | 9201   |
| 1891-1900  | Netherlands | 26758  |
|------------+-------------+--------|
```

现在，我们再一次使用 Rio，不过是通过一个用于构建 ggplot2 可视化的表达式：

```
$ < data/immigration-long.csv Rio -ge 'g + geom_bar(aes(country, count,'\
> ' fill=Period), stat="identity") + scale_fill_brewer(palette="Set1") '\
> '+ labs(x="Country of origin", y="Immigration by decade", title='\
> '"Immigration from Northern Europe\n(columstacked histogram)")' | display
```

-g 选项表示 Rio 应该读取 ggplot2 包。它的输出是一个 PNG 格式的图片（图 7-2）。你可以通过 display 来浏览这个 PNG 图片，其中 display 是 ImageMagick（ImageMagick Studio LLC，2009）的一部分，或者你也可以把输出定向到一个 PNG 文件里。如果你在一个远程终端上，可能看不到任何图形。一个解决方法是在一个特定的目录下启动一个 Web 服务器：

```
$ python -m SimpleHTTPServer 8000
```

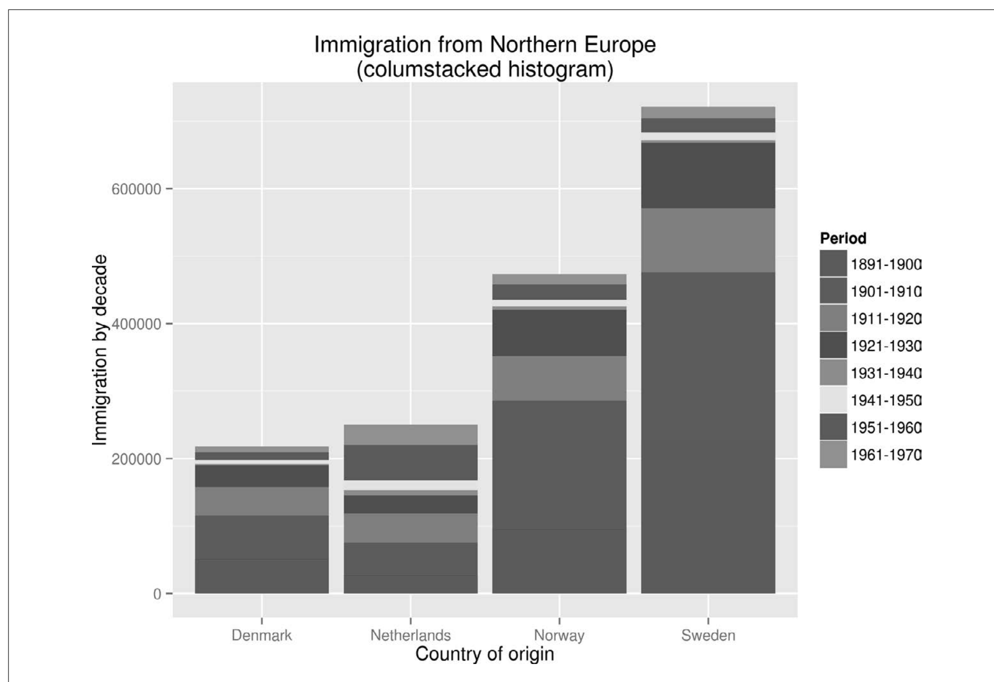


图 7-2：用 Rio 和 ggplot2 绘制的移民量图

确定你可以访问这个端口（这里是 8000）。如果你把 PNG 图片保存在启动 Web 服务器的目录下，就可以通过浏览器在 http://localhost:8000/file.png 访问到这个图片。

7.4.3 直方图

用 Rio（图 7-3）：

```
$ < data/tips.csv Rio -ge 'g+geom_histogram(aes(bill))' | display
```

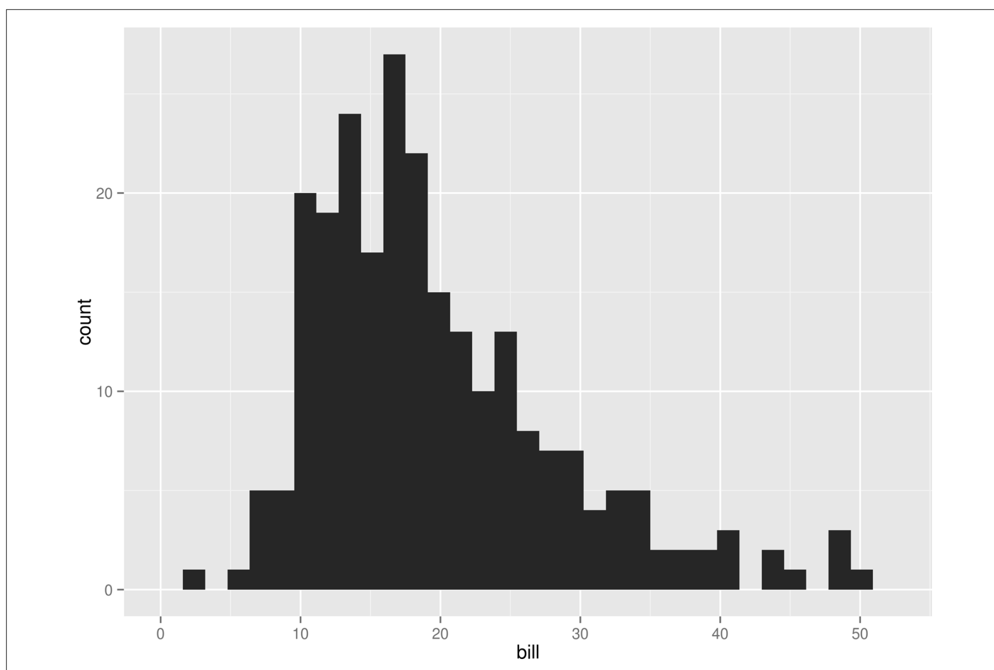
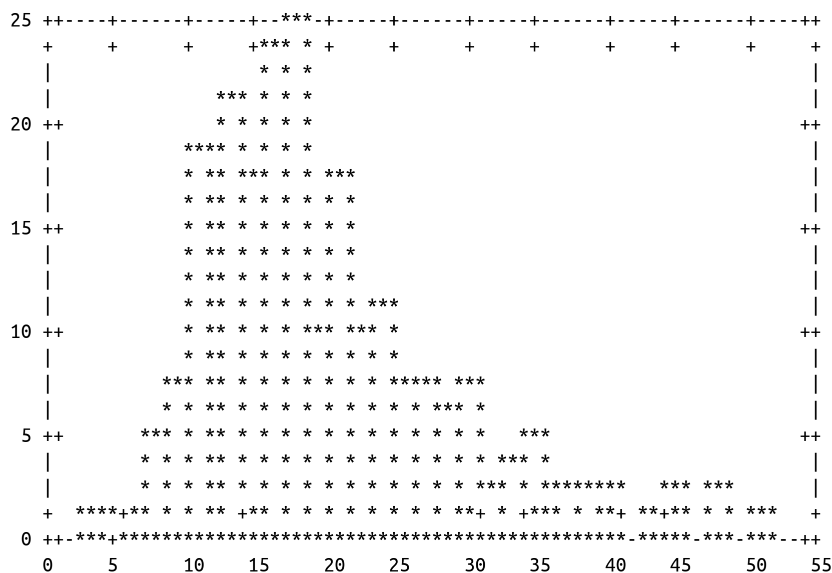


图 7-3：直方图

用 `feedgnuplot`：

```
$ < data/tips.csv csvcut -c bill | feedgnuplot --terminal 'dumb 80,25' \
> --histogram 0 --with boxes --ymin 0 --binwidth 1.5 --unset grid --exit
```



7.4.4 条形图

用 Rio（图 7-4）：

```
$ < data/tips.csv Rio -ge 'g+geom_bar(aes(factor(size)))' | display
```

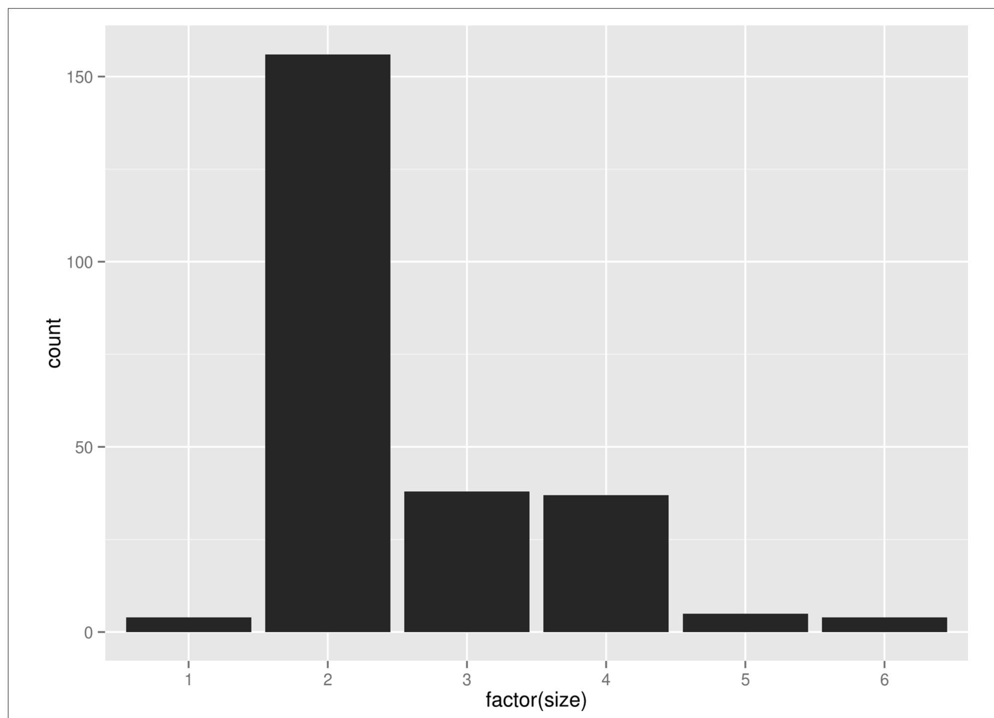
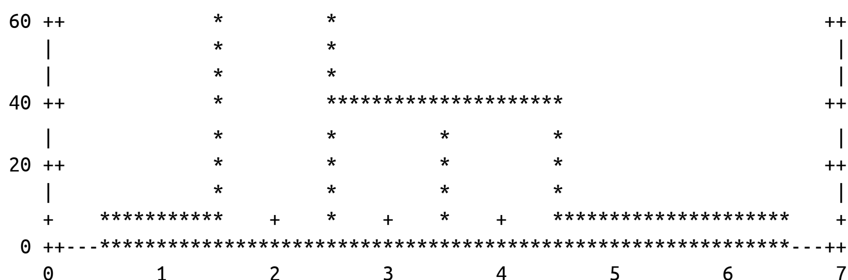


图 7-4：条形图

用 feedgnuplot：

```
$ < data/tips.csv | csvcut -c size | header -d | feedgnuplot --terminal \
> 'dumb 80,25' --histogram 0  --with boxes --unset grid --exit

  160 ++--------+----***********----+---------+---------+---------+--------++
      +         +    *    +    *    +         +         +         +         +
  140 ++             *         *                                           ++
      |              *         *                                            |
      |              *         *                                            |
  120 ++             *         *                                           ++
      |              *         *                                            |
  100 ++             *         *                                           ++
      |              *         *                                            |
      |              *         *                                            |
   80 ++             *         *                                           ++
      |              *         *                                            |
```



7.4.5 密度图

用 Rio（图 7-5）：

```
$ < data/tips.csv Rio -ge 'g+geom_density(aes(tip / bill * 100, fill=sex), '\
> 'alpha=0.3) + xlab("percent")' | display
```

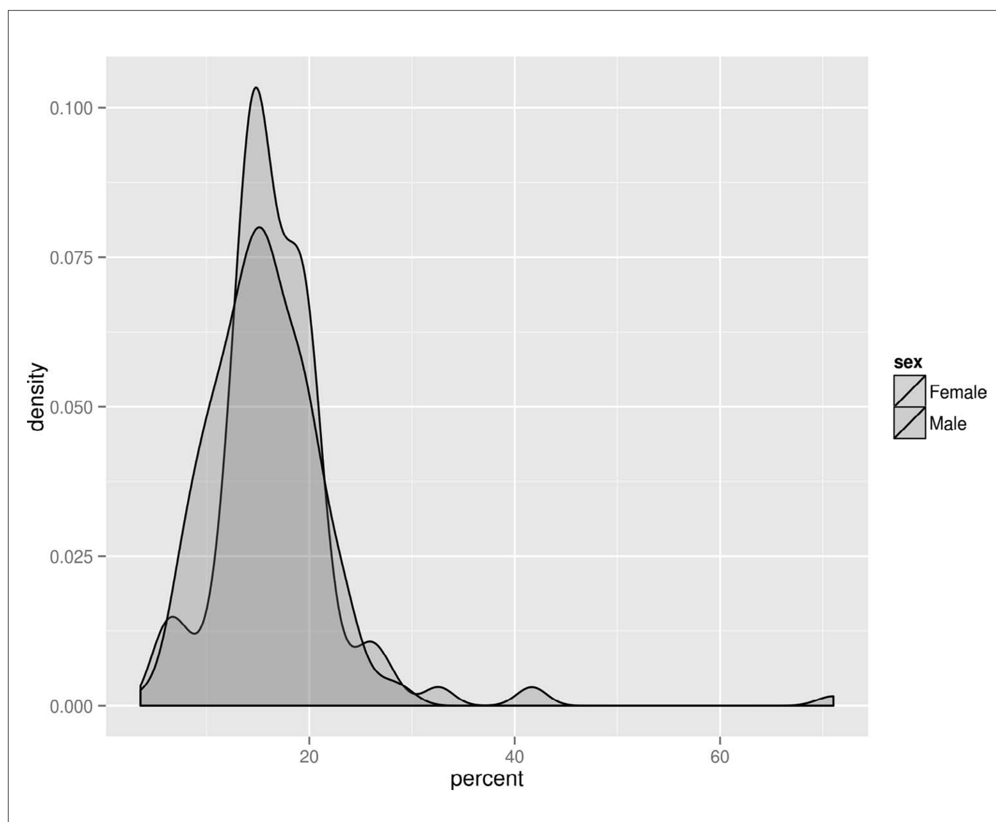


图 7-5：密度图

Feedgnuplot 无法生成密度图，所以最好只生成一个直方图。

7.4.6　箱线图

用 Rio（图 7-6）：

```
$ < data/tips.csv Rio -ge 'g+geom_boxplot(aes(time, bill))' | display
```

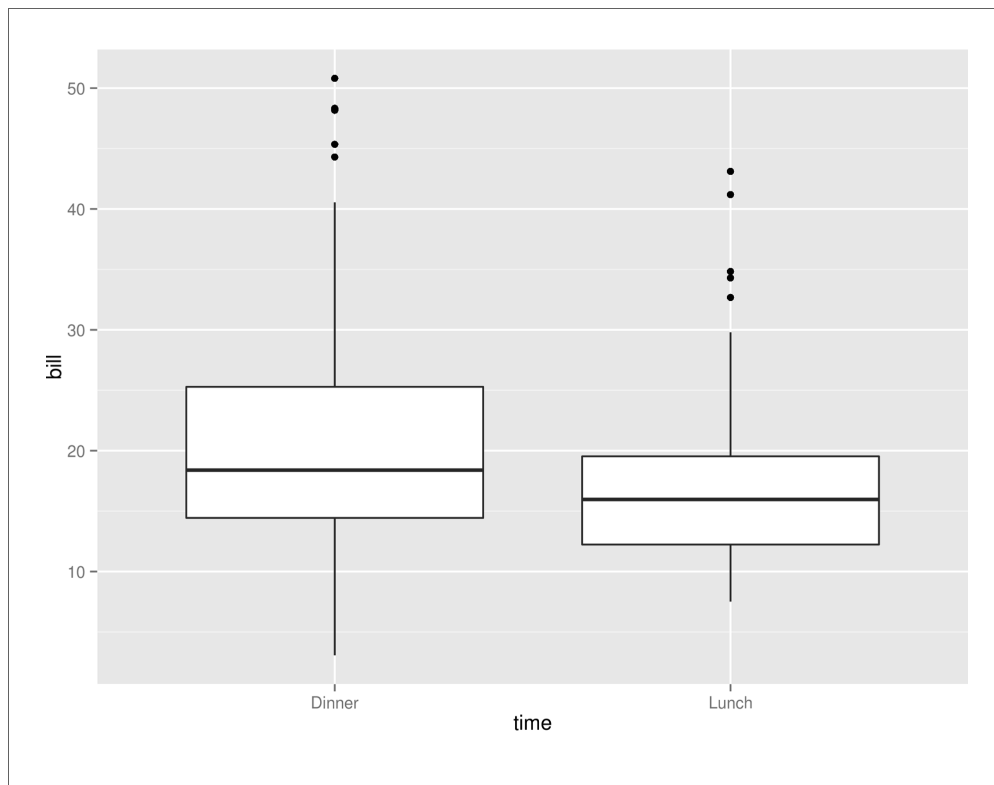


图 7-6：箱线图

不巧的是，无法用 feedgnuplot 绘制箱线图。

7.4.7　散点图

用 Rio（图 7-7）：

```
$ < data/tips.csv Rio -ge 'g+geom_point(aes(bill, tip, color=time))' | display
```

图 7-7：散点图

用 feedgnuplot：

```
$ < data/tips.csv csvcut -c bill,tip | tr , ' ' | header -d | feedgnuplot \
> --terminal 'dumb 80,25' --points --domain --unset grid --exit --style pt 14
```

```
10 ++----+------+-----+------+-----+------+-----+------+-----+------+A---++
   +     +      +     +      +     +      +     +      +     +      +     +
 9 ++                                                            A       ++
   |                                                                      |
 8 ++                                                                    ++
   |                                                  A                   |
   |                                                                      |
 7 ++                                          A                 A       ++
   |                             A     A                                  |
 6 ++                              A    A    A                           ++
   |                             A        A                               |
 5 ++      A                 A A   A A   AA A  AA      A  A     A       ++
   |                              A       A    A     A                  |
 4 ++          A     A  AAAA AAA A  A A  A          A                    ++
   |                AAAAA AAA AA            A             A           |
   |              A  AAAAAAA AA A A  AA   A AA                            |
 3 ++           A   AAAAAAAAAAA A A    AA           AA A                 ++
   |              AAAAAAA AA  A A A     A                   A             |
```

```
2 ++        AA AAAAAAAAA A  A  A AA  A A A                        ++
  +     +   AAAAAAAA +A   AA+     + A     +     +      +     +      +
1 ++--A-+A-A---+--AA-+--A---+-----+------+--A--+-----+-----+-----+----++
  0     5      10     15     20     25     30     35     40     45     50     55
```

7.4.8 折线图

用 Rio（图 7-8）：

```
$ < data/immigration-long.csv Rio -ge 'g+geom_line(aes(x=Period, '\
> 'y=Count, group=Country, color=Country)) + theme(axis.text.x = '\
> 'element_text(angle = -45, hjust = 0))' | display
```

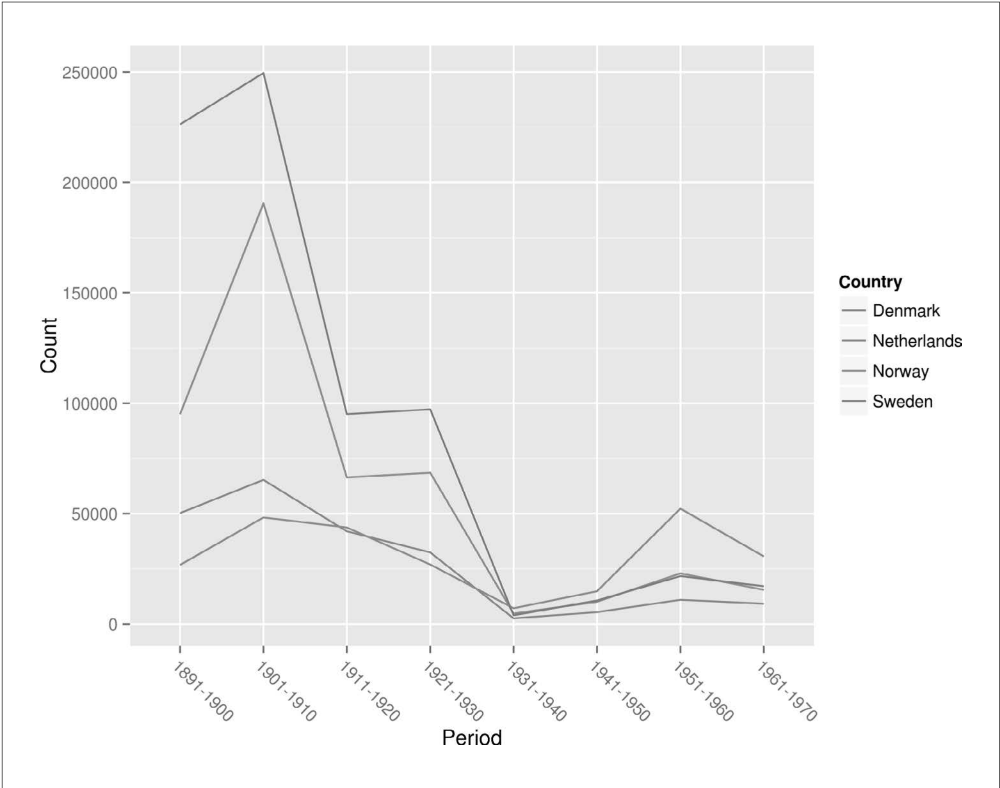
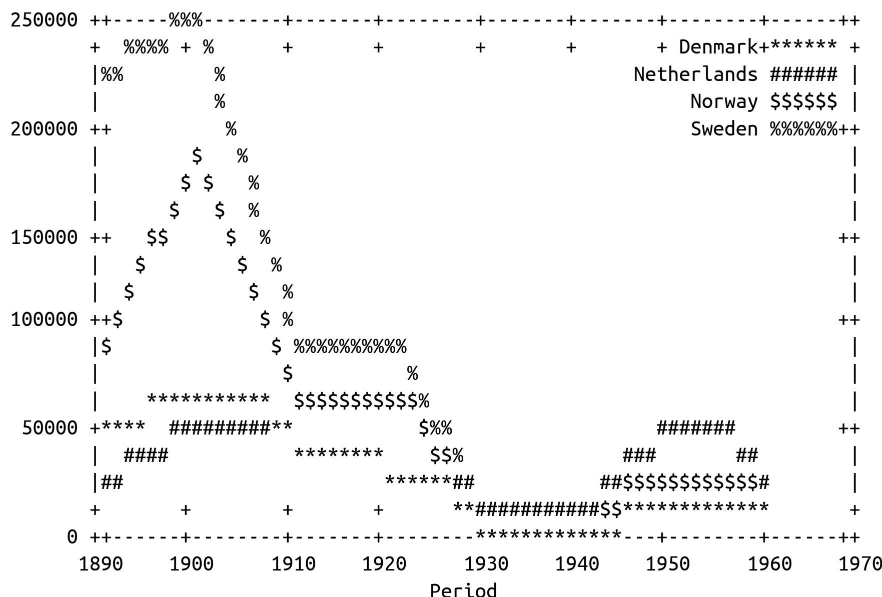


图 7-8：折线图

用 feedgnuplot：

```
$ < data/immigration.csv csvcut -c Period,Denmark,Netherlands,Norway,Sweden |
> header -d | tr , ' ' | feedgnuplot --terminal 'dumb 80,25' --lines \
> --autolegend --domain --legend 0 "Denmark" --legend 1 "Netherlands" \
> --legend 2 "Norway" --legend 3 "Sweden" --xlabel "Period" --unset grid --exit
```



7.4.9 总结

含有 `ggplot2` 的 `Rio` 和含有 Gunplot 的 `feedgnuplot` 都有其优点。用 `Rio` 生成的图形的质量明显高很多。而 `ggplot2` 提供了一种一致而简明的语法，对命令行非常友好。唯一的缺点是，它的输出无法在命令行中直接看到。而这也就是 `feedgnuplot` 派上用场的地方。每个图形具有大致相同的命令行参数。因此，直接生成一小段 Bash 脚本就可以更加容易地在命令行中或者为命令行生成图形。毕竟，命令行的分辨率很低，我们并不要求很大的灵活性。

7.5 延伸阅读

- Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer.
- Janert, P. K. (2009). *Gnuplot in Action*. Manning Publications.
- Tukey, J. W. (1977). *Exploratory Data Analysis*. Pearson.

第 8 章

并行管道

在前面的章节中，我们已经讨论过一次性处理整个任务的命令和管道。然而在实践中，你可能会发现，在要处理的任务中需要多次运行同样的命令或管道。例如，你可能需要：

- 清洗成百上千的网页
- 进行很多次 API 调用，并对它们的输出进行转换
- 在一系列参数值中训练分类器
- 为数据集里的两两特征生成散点图

这些例子中存在某种形式的重复。你可以在习惯的脚本或编程语言中，用 `for` 循环或者 `while` 循环来处理它们。在命令行里，你倾向于做的第一件事是按向上键（退到前一个命令上），修改命令（如果必要的话），并按下回车键（再次运行命令）。这样做两三次还好，但是设想一下对很多文件都这样做的情景。这种方法立刻变得笨重而低效。不过好消息是，我们也可以在命令行里写 `for` 和 `while` 循环。

有时，一个接一个（串行）重复执行快捷命令是高效的。当你有多个核（甚至可能是多个机器）的时候，如果能利用它们就太好了，特别是当你面对的是一个数据密集型任务时。当你使用多个核或多个机器的时候，总体运行时间可能会显著减少。本章，我们将会介绍一个功能非常强大的、正是解决这个问题的工具，叫作 GNU Parallel。GNU Parallel 能让我们对一系列参数使用命令或管道，如数字、文本行以及文件。而且，它允许我们并行执行命令。

8.1 概述

作为穿插的一章，本章讨论了需要多次执行命令行和管道的任务的几种加速方法。本章的主要目标是用实例说明 GNU Parallel 工具的灵活性和强大功能。由于这个工具可以和本书里讨论过的其他任何工具组合在一起，因此它肯定会改变你在数据科学工作中使用命令行的方式。本章你将会学习：

- 在一系列数字、文本行和文件中串行运行命令行
- 用 GNU Parallel 并行运行管道
- 将管道分发到多个机器上

8.2 串行处理

在深入讨论并行处理之前，我们来看看串行循环。我们值得花些时间了解一下它是怎么做的，因为这个功能一直都存在，它的语法很像其他语言里的循环，它会让你真心感激 GNU Parallel 工具。

从本章开头提到的例子中，我们提炼出了需要进行遍历的三类数据：数字、文本行和文件。这三种类型的数据将会分别在下面三个小节中讨论。

8.2.1 对数字进行遍历

设想一下，我们需要计算 0 到 100 中每个偶数的平方。这里有一个工具叫作 `bc`，它基本上是一个在命令行中将公式输入到管道的计算器。计算 4 的平方的命令如下所示：

```
$ echo "4^2" | bc
16
```

对于一次性的计算，这种方式毫无问题。但如果像简介中所提及的那样，我们就需要疯狂地按向上键，改变数值，然后按回车键 51 次！在这种情况下，更好的方法是让 Bash 通过 `for` 循环来为我们处理这项麻烦的工作：

```
$ for i in {0..100..2} ❶
> do
> echo "$i^2" | bc      ❷
> done | tail           ❸
6724
7056
7396
7744
8100
8464
8836
```

```
9216
9604
10000
```

同时在这里还有一些其他的事情：

❶ Bash 有一个特性叫作括号扩展，它会将 `{0..100..2}` 变成一列由空格分隔的数字：`0 2 4 … 98 100`。

❷ 在第一个循环里将变量 `i` 赋值为 1，第二个循环里为 2，以此类推。在这个变量的值之前添加美元符号（`$`），就可以在命令行里使用该变量的值。在执行 `echo` 之前，Shell 会把 `$i` 转换成它的值。注意在 `do` 和 `done` 之间可以有多个命令。

❸ 我们将 `for` 循环的输出通过管道输入到 `tail` 中，以便只看到最后的 10 行数值。

尽管这些语法和你习惯的编程语言相比看起来有些奇怪，它们却是值得记住的，因为它们在 Bash shell 中总会出现。我们马上就会介绍一种更好、更灵活的重复执行命令的方式。

8.2.2　对行进行遍历

第二类用于遍历的数据是行。这些行可以来自于一个文件或者来自于标准输入。这是一个非常通用的方法，因为文本行可以包含任何东西，包括数字、日期、邮箱地址。

想象一下，我们想要给客户发送电子邮件。让我们用 Random User Generator API（http://randomuser.me/）生成一些伪用户：

```
$ cd ~/book/ch08
$ curl -s "http://api.randomuser.me/?results=5" > data/users.json
$ < data/users.json jq -r '.results[].user.email'> data/emails.txt
$ cat data/emails.txt
kaylee.anderson64@example.com
arthur.baker92@example.com
chloe.graham66@example.com
wyatt.nelson80@example.com
peter.coleman75@example.com
```

我们用一个 `while` 循环对 emails.txt 中的文本行进行遍历：

```
$ while read line                                    ❶
> do
> echo "Sending invitation to ${line}."              ❷
> done < data/emails.txt                             ❸
sending invitation to kaylee.anderson64@example.com.
sending invitation to arthur.baker92@example.com.
sending invitation to chloe.graham66@example.com.
sending invitation to wyatt.nelson80@example.com.
sending invitation to peter.coleman75@example.com.
```

❶ 在这里我们使用 `while` 循环，这是因为 Bash 事先并不知道输入中包含多少行。

❷ 尽管在这种情况下 line 变量周围的大括号是不必要的（因为变量名中不能包含句点），但这样做仍然是个好习惯。

❸ 重定向也可以放在 while 之前。

你还可以通过指定特殊文件标准输入 /dev/stdin，交互式地将输入传递给 while 循环。做完之后按 Ctrl+D 键：

```
$ while read i; do echo "You typed: $i."; done < ev/stdin
one
you typed: one.
two
you typed: two.
three
you typed: three.
```

然而，这个方法有一个缺点，就是一旦你按了回车键，对于这行输入，do 和 done 之间的命令将会立即执行。

8.2.3 对文件进行遍历

在本节中，我们将讨论第三种经常需要进行遍历的数据：文件。我们使用 globbing（路径名扩展）而不是 ls，进行特殊字符处理：

```
$ for filename in *.csv
> do
> echo "Processing ${filename}."
> done
Processing countries.csv.
```

如同对数字中的括号进行扩展一样，*.csv 文件在进行 for 循环处理之前，首先会被扩展成一个列表。一种更好的寻找文件的方法是使用 find（Youngman，2008），它：

- 允许详细搜索文件的属性，如大小、访问时间以及权限。
- 处理破折号。
- 处理特殊字符，如空格和换行符。

```
$ find data -name '*.csv' -exec echo "Processing {}" \;
Processing data/countries.csv
Processing data/movies.csv
Processing data/top250.csv
```

这里的例子和之前一样，但现在使用 parallel：

```
$ find data -name "*.csv" -print0 | parallel -0 echo "Processing {}"
Processing data/countries.csv
Processing data/movies.csv
Processing data/top250.csv
```

`-print0` 选项允许处理 `find` 输出的程序正确解释包含有换行符或其他类型空白字符的文件名。如果你完全确定这些文件名并不包含特殊字符，如空格和回车，那么你可以省略 `-print0` 和 `-0` 选项。



如果待处理的列表过于复杂，你可以把结果保存到一个临时文件中，然后使用这个方法对文件中的行进行遍历。

8.3　并行处理

假设我们有一个运行时间非常长的命令，如示例 8-1 所示。

示例 8-1　~/book/ch08/slow.sh

```
#!/bin/bash
echo "Starting job $1"
duration=$((1+RANDOM%5))                    ❶
sleep $duration
echo "Job $1 took ${duration} seconds"
```

❶ `$RANDOM` 是一个内置的 Bash 函数，它可以返回一个 0 到 32 767 之间的伪随机数。对这个数除以 5 取余，然后加 1，就可以确保它在 1 到 5 之间。

这个过程并不会占用所有资源。碰巧我们需要运行这个命令很多次。例如，我们需要下载一个很长的文件序列。

一种原始的并行方法是在后台运行这些命令：

```
$ for i in {1..4}; do
> (./slow.sh $i; echo Processed $i) &   ❶
> done
[1] 3334
[2] 3335
[3] 3336
[4] 3338
Starting job 2
Starting job 1
Starting job 3
Starting job 4
Job 4 took 1 seconds
Processed 4
Job 3 took 4 seconds
Job 2 took 4 seconds
Processed 3
Processed 2
Job 1 took 4 seconds
Processed 1
```

❶ 括号会创建一个子 shell 进程。符号（`&`）确保它在后台运行。

子 shell 进程的问题在于，它们会被立即执行。没有控制最大进程个数的机制。不建议你这样使用：

```
$ while read i; do
> (./slow.sh "$i"; ) &
> done < data/movies.txt
[1] 3404
[2] 3405
[3] 3406
Starting job Star Wars
Starting job Matrix
Starting job Home Alone
[4] 3407
[5] 3410
Starting job Back to the Future
Starting job Indiana Jones
Job Home Alone took 2 seconds
Job Matrix took 2 seconds
Job Star Wars took 2 seconds
Job Back to the Future took 3 seconds
Job Indiana Jones took 4 seconds
```



并不是任何事情都可以并行化：API 调用会限制一定次数，或者一些命令只能有一个实例。



引号很重要。如果我们不把 `$i` 引起来，那么每个电影就只有第一个单词会传进 slow.sh 脚本。

这个原始方法有两个问题。第一个问题是没法控制并发进程的数量。第二个问题是日志记录；某个输出应该是哪个输入的。

8.3.1 GNU Parallel介绍

GNU Parallel（Tange，2014）是一个命令行工具，它让我们可以并行处理命令和管道。这个工具的美妙之处在于，既有工具仍然可以按照原来的方式使用，并不需要修改。在我们深入 GNU Parallel 的细节之前，这里有一个小例子，向你展示一下将上述 `for` 循环并行化是多么容易：

```
$ seq 5 | parallel "echo {}^2 | bc"
1
```

```
4
9
16
25
```

这是 `parallel` 最简单的形式：没有任何选项。正如你所看到的那样，它基本上跟 `for` 循环一样（我们后续就会解释到底发生了什么）。GNU Parallel 通过 110 多个选项（!），提供了很多附加功能。别担心，到本章的的末尾，你将会对其中绝大多数重要选项有一个扎实的理解。

通过运行如下命令安装 GNU Parallel：

```
$ wget http://ftp.gnu.org/gnu/parallel/parallel-latest.tar.bz2
$ tar -xvjf parallel-latest.tar.bz2 > extracted-files
$ cd $(head -n 1 extracted-files)
$ ./configure && make && sudo make install
```



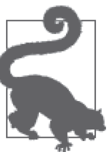
你可能已经注意到，我们一直都在写 GNU Parallel。这是因为有两个工具都以 parallel 命名。如果你使用了数据科学工具箱，那么你已经安装了正确的那个。否则，你可以通过运行 `parallel --version`，仔细检查一下是否已经安装了正确的工具。

你可以验证一下你已经正确安装了 GNU Parallel：

```
$ parallel --version | head -n 1
GNU parallel 20140622
```

要删掉创建的文件和目录，运行如下命令：

```
$ cd ..
$ rm -r $(head -n 1 extracted-files)
$ rm parallel-latest.tar.bz2 extracted-files
```



如果你像我们一样经常使用 `parallel`，那你可能会希望创建一个别名（例如，`p`），将 `alias p=parallel` 加入到你的 ~/.bashrc 里即可。（在本章里，为清楚起见，我们只使用 `parallel`。）

8.3.2 指定输入

GNU Parallel 中最重要的参数就是那些要对每个输入都执行的命令。但问题是：输入项应该插入到命令行的哪个位置呢？如果你不指定任何东西，输入项将会追加到命令的后面。虽然这通常是你想要的效果，但最好可以通过一个或多个占位符，明确指定输入项应该插入到命令的什么地方。



为 GNU Parallel 提供输入的方式有很多。我们倾向于用管道处理输入（如同在本章里做的那样），因为这种方法是普遍适用的，允许我们构建一个从左到右的管道。在 `parallel` 的操作说明页面可以找到提供输入的其他方式。

在大多数情况下，你可能想要使用整个输入项。对此，你只需要一个占位符即可。你可以用两个大括号（`{}`）来指定占位符：

```
$ seq 5 | parallel echo {}
```

当输入项是一个文件时，你可以使用一些特殊占位符来修改文件名。例如，通过 `{./}`，只有主文件名会被使用。

如果输入行里含有多个用分隔符分隔的部分，那你可以在占位符里加入数字。例如：

```
$ < input.csv parallel -c, "mv {1} {2}"
```

这里你可以使用同样的占位符修饰符，还可以重用相同的输入项。如果 `parallel` 的输入是一个含有文件头的 CSV 文件，那么你可以把列名当作占位符：

```
$ < input.csv parallel -c, --header : "invite {name} {email}"
```

有时你只想运行相同的命令，而不改变输入。这在 `parallel` 里面也是可行的。我们只需要指定 `-N0` 选项，其输入就会作为你要执行的行数。

```
$ seq 5 | parallel -N0 "echo The command line rules"
The command line rules
The command line rules
The command line rules
The command line rules
The command line rules
```



如果你想知道 GNU Parallel 命令是否设置正确，可以加上 `--dryrun` 选项。GNU Parallel 会打印出所有命令，如同它们已经被执行了一样，却非真正执行这些命令。

8.3.3 控制并发任务的个数

Parallel 默认在每个 CPU 核上并发运行一个任务。你可以通过 `--jobs` 或 `-j` 选项控制要并发运行的任务个数。简单指定一个数字，例如 n，意味着 n 个任务将会并行运行。如果你在 n 前面放一个加号，那么 `parallel` 将会运行 $m+n$ 个任务，其中 m 是 CPU 核的个数。如果你在 n 之前放一个减号，那么 `parallel` 将会运行 $m-n$ 个任务。你还可以对 `-j` 选项指定一个百分比。其默认值是 100% 的 CPU 核数。最优的并发任务数取决于运行的实际命令：

```
$ seq 5 | parallel -j0 "echo Hi {}"
Hi 1
Hi 2
Hi 3
Hi 4
Hi 5

$ seq 5 | parallel -j200% "echo Hi {}"
Hi 1
Hi 2
Hi 3
Hi 4
Hi 5
```

如果你设置 `-j1`，那么命令会串行执行。尽管这和工具名称并不相符，它仍然有其用处。例如，当你需要访问一个 API 的时候，它的每次调用只允许有一个连接。如果你设置 `-j0`，那么 `parallel` 会执行尽可能多的并行任务。这与循环执行子 shell 相似，并不推荐。

8.3.4 记录日志和输出

要存储每个命令的输出，你可能会尝试做以下事情：

```
$ seq 5 | parallel "echo \"Hi {}\" > data/hi-{}.txt"
```

这会把输出保存到各个文件里。或者，如果你想要把所有东西都保存到一个大文件里，可以按这样来做：

```
$ seq 5 | parallel "echo Hi {}" >> data/one-big-file.txt
```

然而，GNU Parallel 提供了 `--results` 选项，它会把每个任务的输出存入一个单独的文件里，其文件名取决于输入值：

```
$ seq 5 | parallel --results data/outdir "echo Hi {}"
Hi 1
Hi 2
Hi 3
Hi 4
Hi 5
$ find data/outdir
data/outdir
data/outdir/1
data/outdir/1/1
data/outdir/1/1/stderr
data/outdir/1/1/stdout
data/outdir/1/3
data/outdir/1/3/stderr
data/outdir/1/3/stdout
data/outdir/1/5
data/outdir/1/5/stderr
data/outdir/1/5/stdout
```

```
data/outdir/1/2
data/outdir/1/2/stderr
data/outdir/1/2/stdout
data/outdir/1/4
data/outdir/1/4/stderr
data/outdir/1/4/stdout
```

当你并行执行多个任务时，任务执行的顺序可能跟输入的顺序并不对应。因此任务的输出也是混在一起的。要保持同样的顺序，指定 `--keep-order` 或 `-k` 选项即可。

记录哪个输入生成了哪个输出有时也是有用处的。GNU Parallel 允许你通过 `--tag` 选项为输出打标签：

```
$ seq 5 | parallel --tag "echo Hi {}"
1       Hi 1
2       Hi 2
3       Hi 3
4       Hi 4
5       Hi 5
```

8.3.5　创建并行工具

在本章开头使用过的 `bc` 工具，本身并不是并行的。然而，我们可以用 `parallel` 使之并行化。数据科学工具箱里包含一个叫作 `pbc`（Janssens，2014）的工具。它的源码在示例 8-2 中。

示例 8-2　并行化 bc（pbc）

```
#!/usr/bin/env bash
parallel -C, -k -j100% "echo '$1' | bc -l"
```

这个工具也能让我们简化本章开头所使用的代码：

```
$ seq 100 | pbc '{1}^2' | tail
8281
8464
8649
8836
9025
9216
9409
9604
9801
10000
```

这个工具的工作原理如下。你可能还记得 `seq 100` 会生成 1 到 100 的整数，每行一个数。这些行通过管道传给 `pbc`，它会相应地传递给 `parallel`。传到 `{1}` 里的参数在发送到 `bc` 之前会由 `parallel` 进行处理。这意味着，`{1}` 会被行中第一列的值取代（只有一个列）。

8.4 分布式处理

有时你需要比你本地机器所有核都更为强大的能力。幸好，GNU Parallel 还可以利用远程机器的力量，让我们对管道进行加速。

GNU Parallel 的优点在于，它并不需要安装在远程机器上。它只要求你能通过 SSH 连接远程机器，这也是 GNU Parallel 分发管道的途径。（远程安装 GNU Parallel 是有帮助的，因为它可以确定每台远程机器上有多少核可以利用；后续有更多介绍。）

首先，我们会获得运行中的 AWS EC2 实例列表。如果你没有任何远程机器，别担心，你可以用 `--sshlogin :` 替换所有出现的 `--slf instances` 来达到，它会告诉 GNU Parallel 使用哪些远程机器。通过这种方式，你仍然可以跟上本节里的示例。

一旦知道接管了哪些远程机器，我们就会考虑三类分布式处理方式：

- 在远程机器上运行一般命令。
- 在远程机器间直接分发本地数据。
- 将文件发送到远程机器上进行处理，然后获取结果。

8.4.1 获得运行中的AWS EC2实例列表

在本节里，我们创建一个名为 instances 的文件，其中每行都包含一个远程机器的域名。我们将在本节使用亚马逊 Web 服务。如果你使用不同的云计算服务，或者你有自己的服务器，那么要确保你已经自行创建了一个 instances 文件。

你可以通过 `aws` 从命令行中获得运行中的 AWS EC2 实例列表。它是访问 AWS API（Amazon Web Services，2014）的命令行接口。如果你并没有使用数据科学工具箱，那么按照如下方式用 `pip`（PyPA，2014）安装 `awscli`：

```
$ pip install awscli
```

通过 `aws`，你几乎可以完成在线上 AWS 管理控制台中所能做的所有事情。我们只是用这个工具从 AWS 里获得运行中的 EC2 实例列表，但它还可以做更多的事。我们假设你已经知道如何启动实例，通过线上 AWS 管理控制台，或者通过 `aws` 命令行工具。

命令 `aws ec2 describe-instances` 会返回很多关于 EC2 实例的信息，并以 JSON 格式存储（见 AWS 文档，http://docs.aws.amazon.com/cli/latest/reference/ec2/describe-instances.html）。我们用 `jq` 将相关字段提取出来：

```
$ aws ec2 describe-instances | jq '.Reservations[].Instances[] | '\
> '{public_dns: .PublicDnsName, state: .State.Name}'
{
  "state": "running",
```

```
  "public_dns": "ec2-54-88-122-140.compute-1.amazonaws.com"
}
{
  "state": "stopped",
  "public_dns": null
}
```

EC2 实例可能的状态有等待中、运行中、正在关闭、已结束、正在关停和已关停。由于只能把管道分发到运行中的实例里，所以我们将未运行的实例过滤掉：

```
$ aws ec2 describe-instances | jq -r '.Reservations[].Instances[] | '\
> 'select(.State.Name=="running") | .PublicDnsName' > instances
$ cat instances
ec2-54-88-122-140.compute-1.amazonaws.com
ec2-54-88-89-208.compute-1.amazonaws.com
```

（如果我们漏掉了代表 raw 的 `-r`，那么域名两边会有双引号。）我们将输出保存到 instances 中，以便后续可以把它传递给 `parallel`。

前面提到过，`parallel` 使用 SSH 连接 EC2 实例。将下面这些加入到 ~/.ssh/config 中，使得 SSH 知道如何连接 EC2 实例：

```
Host *.amazonaws.com
    IdentityFile ~/.ssh/MyKeyFile.pem
    User ubuntu
```

你的用户名可能跟 `ubuntu` 不同，这取决于你正在使用的版本。

8.4.2　在远程机器上运行命令

第一种分布式处理方式是在远程机器上运行一般命令。让我们通过运行命令行工具 `hostname` 得到域名列表，来仔细检查一下 `parallel` 是否工作：

```
$ parallel --nonall --slf instances hostname
ip-172-31-23-204
ip-172-31-23-205
```

这里 `--slf` 选项的全称是 `--sshloginfile` 选项。`--nonall` 选项让 `parallel` 在每台远程机器的 instances 文件上执行相同的命令，而不使用任何参数。记住，如果你没有任何远程机器可用，可以将 `--slf instances` 替换为 `--sshlogin :`，使得命令行可以在你的本地机器上运行。

```
$ parallel --nonall --sshlogin : hostname
data-science-toolbox
```

在每台远程机器上运行一次相同的命令只需要每台机器上有一个核。如果想要把传入的一列参数分发给 `parallel`，它就可能使用多个 CPU 核。如果没有明确指定 CPU 核的个数，

那么 parallel 将会尝试确定它：

```
$ seq 2 | parallel --slf instances echo 2>1 | fold
bash: parallel: command not found
parallel: Warning: Could not figure out number of cpus on ec2-54-88-122-140.comp
ute-1.amazonaws.com (). Using 1.
1
2
```

在这种情况下，我们将 parallel 安装在两台远程机器中的一台上。我们会得到一个警告信息，说在其中一台机器上没有发现 parallel。结果，parallel 无法确定核的个数，它就会默认使用一个核。当你收到这个警告信息时，可以做以下四件事之一。

- 别担心，并且乐于在每台机器使用一个核。
- 通过 -j 选项指定每台机器上的任务个数。
- 指定每台机器上所使用的核数，例如，把 2/（如果你想要两个核）放在 instances 文件中每个域名的前面。

用包管理器安装 GNU Parallel（并不是说它通常不是最新版本）。例如，在 Ubuntu 上：

```
$ parallel --nonall --slf instances "sudo apt-get install -y parallel"
```

8.4.3 在远程机器间分发本地数据

第二种分布式处理方式是直接在远程机器间分发本地数据。假设你有一个非常大的数据集，想要用多个远程机器进行处理。简单起见，我们将要对 1 到 1000 中的所有整数求和。首先，我们打印出远程机器的域名和用 wc 得到的输入长度，来验证一下输入是否已经分发：

```
$ seq 1000 | parallel -N100 --pipe --slf hosts "(hostname; wc -l) | paste -sd:"
ip-172-31-23-204:100
ip-172-31-23-205:100
ip-172-31-23-205:100
ip-172-31-23-204:100
ip-172-31-23-205:100
ip-172-31-23-204:100
ip-172-31-23-205:100
ip-172-31-23-204:100
ip-172-31-23-205:100
ip-172-31-23-204:100
```

我们可以验证，1000 个数字均匀分发到了 100 的子集里（由 -N100 指定）。现在，我们已经做好准备对这些数字求和：

```
$ seq 1000 | parallel -N100 --pipe --slf hosts "paste -sd+ | bc" |
> paste -sd+ | bc
500500
```

这里，我们还可以立即对从远程机器上拿回的 10 个和相加。让我们仔细检查一下结果是否正确：

```
$ seq 1000 | paste -sd+ | bc
500500
```

好的，确实有效。如果你有一个更大的命令要在远程机器上执行，你还可以把它放在一个独立的脚本里面，并用 `parallel` 上传。在这里，我们创建一个简单的命令行工具，叫作 sum：

```
#!/usr/bin/env bash
paste -sd+ | bc
```

别忘记赋予它可执行权限，如第 4 章讨论的那样。下面这个命令首先会上传文件 sum：

```
$ seq 1000 | parallel -N100 --basefile sum --pipe --slf instances './sum' |
> ./sum
500500
```

当然，对 1000 个数字求和只是一个简单例子。在本地做这件事会快得多。然而，我们希望从这个简单例子中可以清晰认识到，GNU Parallel 拥有强大的功能。

8.4.4 在远程机器上处理文件

第三种分布式处理方式是将文件发送到远程机器进行处理，然后获取结果。假设我们想要统计纽约市每个区在 311 收到服务请求的频率。在本地机器上并没有数据，所以我们先用 API 从 NYC Open Data（https://data.cityofnewyork.us/）上获取下来：

```
$ seq 0 100 900 | parallel  "curl -sL 'http://data.cityofnewyork.us/resource'"\
> "'/erm2-nwe9.json?\$limit=100\$offset={}' | jq -c '.[]' | gzip > {#}.json.gz"
```

注意，`jq -c '.[]'` 用于将 JSON 对象数组扁平化，使得每行有一个对象。我们现在有了 10 个含有压缩的 JSON 数据的文件。让我们看看每行 JSON 长什么样：

```
$ zcat 1.json.gz | head -n 1 | fold
{"school_region":"Unspecified","park_facility_name":"Unspecified","x_coordinate_
state_plane":"945974","agency_name":"Department of Health and Mental Hygiene","u
nique_key":"147","facility_type":"N/A","status":"Assigned","school_address":"Uns
pecified","created_date":"2006-08-29T21:25:23","community_board":"01 STATEN ISLA
ND","incident_zip":"10302","school_name":"Unspecified","location":{"latitude":"4
0.62745427115626","longitude":"-74.13789056665027","needs_recoding":false},"comp
laint_type":"Food Establishment","city":"STATEN ISLAND","park_borough":"STATEN I
SLAND","school_state":"Unspecified","longitude":"-74.13789056665027","intersecti
on_street_1":"DECKER AVENUE","y_coordinate_state_plane":"167905","due_date":"200
6-10-05T21:25:23","latitude":"40.62745427115626","school_code":"Unspecified","sc
hool_city":"Unspecified","address_type":"INTERSECTION","intersection_street_2":"
BARRETT AVENUE","school_number":"Unspecified","resolution_action_updated_date":"
2006-10-06T00:00:17","descriptor":"Handwashing","school_zip":"Unspecified","loca
tion_type":"Restaurant/Bar/Deli/Bakery","agency":"DOHMH","borough":"STATEN ISLAN
D","school_phone_number":"Unspecified"}
```

如果我们想要在本地机器上获得每个区的服务请求个数，要运行如下命令：

```
$ zcat *.json.gz |                ❶
> jq -r '.borough' |              ❷
> tr '[A-Z] ' '[a-z]_' |          ❸
> sort | uniq -c |                ❹
> awk '{print $2","$1}' |         ❺
> header -a borough,count |       ❻
> csvsort -rc count | csvlook     ❼
|----------------+--------|
| borough        | count  |
|----------------+--------|
| unspecified    | 467    |
| manhattan      | 274    |
| brooklyn       | 103    |
| queens         | 77     |
| bronx          | 44     |
| staten_island  | 35     |
|----------------+--------|
```

由于这是一个非常长的管道，而且我们会在 parallel 中再次使用它，因此值得仔细看一下它：

❶ 用 zcat 扩展所有的压缩文件。
❷ 对每个请求，用 jq 抽取出区名。
❸ 将区名转成小写字母，并用下划线代替空格（因为 awk 会默认按照空格切分）。
❹ 用 sort 和 uniq 统计每个区的出现次数。
❺ 将 count 和 borough 字段反转，并用 awk 使它们以逗号分隔。
❻ 用 header 添加一个数据头。
❼ 用 csvsort（Groskopf，2014）对 count 排序，并用 cvslook 打印出一个表格。

想象一下，当我们的机器很慢的时候，我们无法在本地运行这个管道。我们可以用 GNU Parallel 在远程机器间分发本地文件，让它们做处理，并且获得结果：

```
$ ls *.json.gz |                                                          ❶
> parallel -v --basefile jq \                                             ❷
> --trc {.}.csv \                                                         ❸
> --slf instances \                                                       ❹
> "zcat {} | ./jq -r '.borough' | tr '[A-Z] ' '[a-z]_' | sort | uniq -c |"\
> " awk '{print \$2\",\"\$1}' > {.}.csv"                                  ❺
zcat 10.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 2.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 1.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 3.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 4.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 5.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 6.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 7.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 8.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
zcat 9.json.gz | ./jq -r '.borough' | sort | uniq -c | awk '{print $2","$1}'
```

这个长命令按如下方式划分：

❶ 用 ls 打印文件列表，并通过管道把它传入 parallel。

❷ 将 jq 二进制文件传输到每台远程机器上（幸运的是，jq 没有任何依赖）。最后这个文件会从远程机器上删掉，因为我们指定了 --trc 选项（它意味着 --cleanup 选项）。

❸ --trc {.}.csv 选项是 --transfer --return {.}.csv --cleanup 的缩写（用输入文件名替换替代符 {.}，而没有最后的扩展）。在这里，这将意味着 JSON 文件被传输到远程机器上，CSV 文件返回到本地机器，每个任务完成之后所有文件都会从远程机器上删除。

❹ 指定域名列表。记住，如果你想要在本地做这件事，可以设置 --sshlogin : 而非 --self instances。

❺ 注意 awk 表达式中的转义。引号有时比较微妙。在这里，美元符号和双引号都进行了转义。如果引号过于混乱，记住，你可以把管道转换为独立的命令行工具，就如同在 sum 中做的那样。

在这个命令运行过程中的某个时刻，如果我们在一台远程机器上运行 ls，将会看到 parallel 确实传递（并清除）了二进制文件 jq、JSON 文件以及 CSV 文件：

```
$ ssh $(head -n 1 instances) ls
1.json.csv
1.json.gz
jq
```

每个 CSV 文件看起来都像这样：

```
$ cat 1.json.csv
bronx,3
brooklyn,5
manhattan,24
queens,3
staten_island,2
unspecified,63
```

我们可以使用 Rio 和 R 中的 aggregate 函数，将每个 CSV 文件中的统计值相加：

```
$ cat *.csv | header -a borough,count |
> Rio -e 'aggregate(count ~ borough, df, sum)' |
> csvsort -rc count | csvlook
|----------------+--------|
| borough        | count  |
|----------------+--------|
| unspecified    | 467    |
| manhattan      | 274    |
| brooklyn       | 103    |
| queens         | 77     |
| bronx          | 44     |
| staten_island  | 35     |
|----------------+--------|
```

或者，如果你更愿意对总计结果使用 SQL，可以使用在第 5 章讨论过的 `csvsql`：

```
$ cat *.csv | header -a borough,count |
> csvsql --query 'SELECT borough, SUM(count) AS count FROM stdin '\
> 'GROUP BY borough ORDER BY count DESC' | csvlook
|----------------+--------|
| borough        | count  |
|----------------+--------|
| unspecified    | 467    |
| manhattan      | 274    |
| brooklyn       | 103    |
| queens         | 77     |
| bronx          | 44     |
| staten_island  | 35     |
|----------------+--------|
```

8.5 讨论

作为数据科学家，我们处理数据，而且有时会面对很多数据。这意味着我们经常需要多次运行某个命令，或者将数据密集型的命令分发到多个 CPU 核或多台机器上。本章已经展示出，命令并行化是很容易的。GNU Parallel 是一个非常强大和灵活的工具，可以对一般命令行工具加速，并将它们分发到多个核和远程机器上。它提供了很多功能，在本章里我们只是触及了皮毛。一些我们没涉及的 GNU Parallel 特性包括：

- 保留所有任务的日志。
- 当一台机器的负载在一定阈值以下时只启动新任务。
- 超时暂停、重启和重试任务。

一旦你对 GNU Parallel 及其最重要的选项有一个基本了解，我们推荐你看看“延伸阅读”中列出的教程。

8.6 延伸阅读

- Tange, O. (2011). GNU Parallel—The Command-Line Power Tool. *;Login: The USENIX Magazine*, 36(1), 42–47. Retrieved from http://www.gnu.org/s/parallel.
- Tange, O. (2014). GNU Parallel Tutorial. Retrieved from http://www.gnu.org/software/parallel/parallel_tutorial.html.
- Amazon Web Services (2014). AWS Command Line Interface Documentation. Retrieved from http://aws.amazon.com/documentation/cli/.

第 9 章

数据建模

在本章中，我们将会进行 OSEMN 模型的第四个步骤（最后一个需要计算机的步骤）：数据建模。通常来说，数据建模是对数据构建一个抽象的或高层次的描述。就像生成可视化时一样，从独立的数据点向回退了一步。

一方面，可视化是以形状、位置和颜色为特征的，我们可以通过视觉解释它们。而另一方面，模型内部以一堆数字为特征，例如计算机能用它们来预测新数据。（我们仍然会对模型进行可视化，以便能够试图理解它们，看看它们是怎样工作的。）

在本章里，我们会考虑四类常见的数据建模算法：

- 降维
- 聚类
- 回归
- 分类

这四类算法源于机器学习领域。因此，我们将会改变一些词汇。假设我们有一个 CSV 文件，又叫作*数据集合*。除了文件头，每一行都会被当作一个*数据点*。为简单起见，我们假设含有数值的每一列都是一个输入*特征*。如果一个数据还含有一个非数值字段，例如 Iris 数据集的 species 列，那它又叫作数据点的*标签*。

前两类算法（降维和聚类）通常是非监督的，这意味着它们只会基于数据集合本身的特征生成模型。后两类算法（回归和分类）从定义上说是监督算法，这意味着它们还会在模型中使用标签。



这并不是机器学习的介绍。这意味着我们必须跳过很多细节。我们强烈建议你在数据中盲目应用算法之前先熟悉一下它们。

9.1 概述

本章你将会学到如何：

- 降低数据集的维数
- 通过三种聚类算法发现数据的分组
- 通过回归预测白酒的品质
- 通过预测 API 把酒分成红酒或白酒

9.2 更多的酒，来吧！

在本章里，我们将使用一个酒品数据集——具体来说，是红色和白色的葡萄牙 Vinho Verde 葡萄酒。每个数据点都代表一种酒，包含 11 个物理化学性质：(1) 固定酸度，(2) 挥发酸性，(3) 柠檬酸，(4) 残余糖分，(5) 氯化物，(6) 游离二氧化硫，(7) 总二氧化硫，(8) 密度，(9) pH 值，(10) 硫酸盐，(11) 酒精。这里还有一个品质分数。这个分数在 0（非常不好）到 10（优秀）之间，它是由酒类专家给出的至少三次评估值的中间值。关于这个数据集的更多信息，在酒品数据集网页（http://archive.ics.uci.edu/ml/datasets/Wine+Quality）中可以找到。

这里有两个数据集：一个是白酒的，一个是红酒的。第一步是用 `curl` 获取这两个数据集（当然也使用了 `parallel`，因为我们没有那么多时间等待）：

```
$ cd ~/book/ch09/data
$ parallel "curl -sL http://archive.ics.uci.edu/ml/machine-learning-databases"\
> "/wine-quality/winequality-{}.csv > wine-{}.csv" ::: red white
```

（3 个冒号是另一种向 `parallel` 传递数据的方式。）让我们用 `head` 来查看数据，用 `wc -l` 来统计行数：

```
$ head -n 5 wine-{red,white}.csv | fold
==> wine-red.csv <==
"fixed acidity";"volatile acidity";"citric acid";"residual sugar";"chlorides";"f
ree sulfur dioxide";"total sulfur dioxide";"density";"pH";"sulphates";"alcohol";
"quality"
7.4;0.7;0;1.9;0.076;11;34;0.9978;3.51;0.56;9.4;5
7.8;0.88;0;2.6;0.098;25;67;0.9968;3.2;0.68;9.8;5
7.8;0.76;0.04;2.3;0.092;15;54;0.997;3.26;0.65;9.8;5
11.2;0.28;0.56;1.9;0.075;17;60;0.998;3.16;0.58;9.8;6
```

```
==> wine-white.csv <==
"fixed acidity";"volatile acidity";"citric acid";"residual sugar";"chlorides";"f
ree sulfur dioxide";"total sulfur dioxide";"density";"ph";"sulphates";"alcohol";
"quality"
7;0.27;0.36;20.7;0.045;45;170;1.001;3;0.45;8.8;6
6.3;0.3;0.34;1.6;0.049;14;132;0.994;3.3;0.49;9.5;6
8.1;0.28;0.4;6.9;0.05;30;97;0.9951;3.26;0.44;10.1;6
7.2;0.23;0.32;8.5;0.058;47;186;0.9956;3.19;0.4;9.9;6
$ wc -l wine-{red,white}.csv
  1600 wine-red.csv
  4899 wine-white.csv
  6499 total
```

第一眼望去这个数据已经非常干净了。我们仍然要把它清洗一下，使它更符合大多数命令行工具的要求。具体说来，我们将会：

- 把文件头转成小写
- 将分号转成逗号
- 将空格转成下划线
- 删掉不必要的引号

这些事情都可以用 tr 处理。我们这次用一个 for 循环（由于历史原因）来处理这两个数据集：

```
$ for T in red white; do
> < wine-$t.csv tr '[A-Z]; ' '[a-z],_' | tr -d \" > wine-${T}-clean.csv
> done
```

我们把两个数据集合并到一起。我们会使用 scvstack 来增加一个叫作 type 的列，对第一个文件中的行赋值 red，对第二个文件中的行赋值 white：

```
$ HEADER="$(head -n 1 wine-red-clean.csv),type"
$ csvstack -g red,white -n type wine-{red,white}-clean.csv |
> csvcut -c $HEADER > wine-both-clean.csv
```

这个新列 type 被加在了表的前面。由于本章里要使用的一些命令行工具会假设类别的标签在最后一列，我们用 csvcut 把各个列重新排列一下。我们在调用 csvstack 之前把需要的文件头临时存储在一个变量 HEADER 中，而不是把所有 13 个列都敲进去。

最好检查一下数据集中是否存在缺失值：

```
$ csvstat wine-both-clean.csv --nulls
  1. fixed_acidity: False
  2. volatile_acidity: False
  3. citric_acid: False
  4. residual_sugar: False
  5. chlorides: False
  6. free_sulfur_dioxide: False
  7. total_sulfur_dioxide: False
```

```
 8. density: False
 9. ph: False
10. sulphates: False
11. alcohol: False
12. quality: False
13. type: False
```

真棒！出于好奇，我们看看红酒和白酒的品质分布是什么样的：

```
$ < wine-both-clean.csv Rio -ge 'g+geom_density(aes(quality, '\
> 'fill=type), adjust=3, alpha=0.5)' | display
```

从图 9-1 的密度图中，我们可以看到白酒的品质分布更倾向于较高的值。

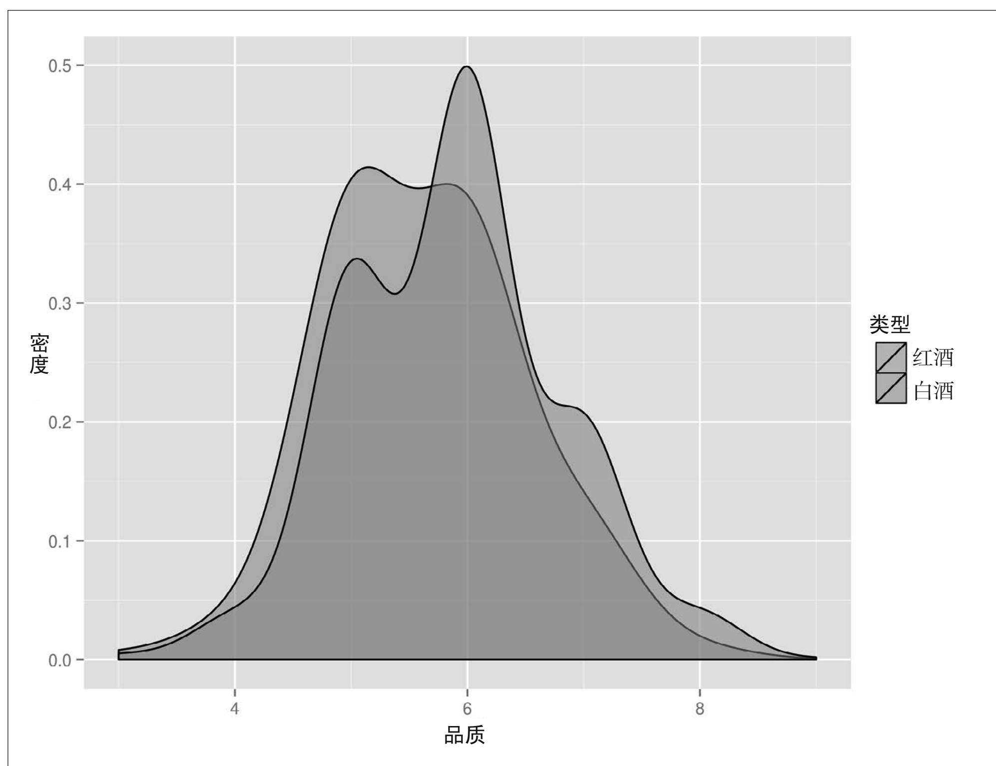


图 9-1：用密度图比较红酒和白酒的品质

这是否意味着，白酒总体来说比红酒更好，或者白酒专家比红酒专家更容易给出高分值？数据并没有告诉我们这些。或者，酒精含量和品质可能会有一定的关联？我们再用 `Rio` 和 `ggplot2` 来查明（参见图 9-2）：

```
$ < wine-both-clean.csv Rio -ge 'ggplot(df, aes(x=alcohol, y=quality, '\
> 'color=type)) + geom_point(position="jitter", alpha=0.2) + '\
> 'geom_smooth(method="lm")' | display
```

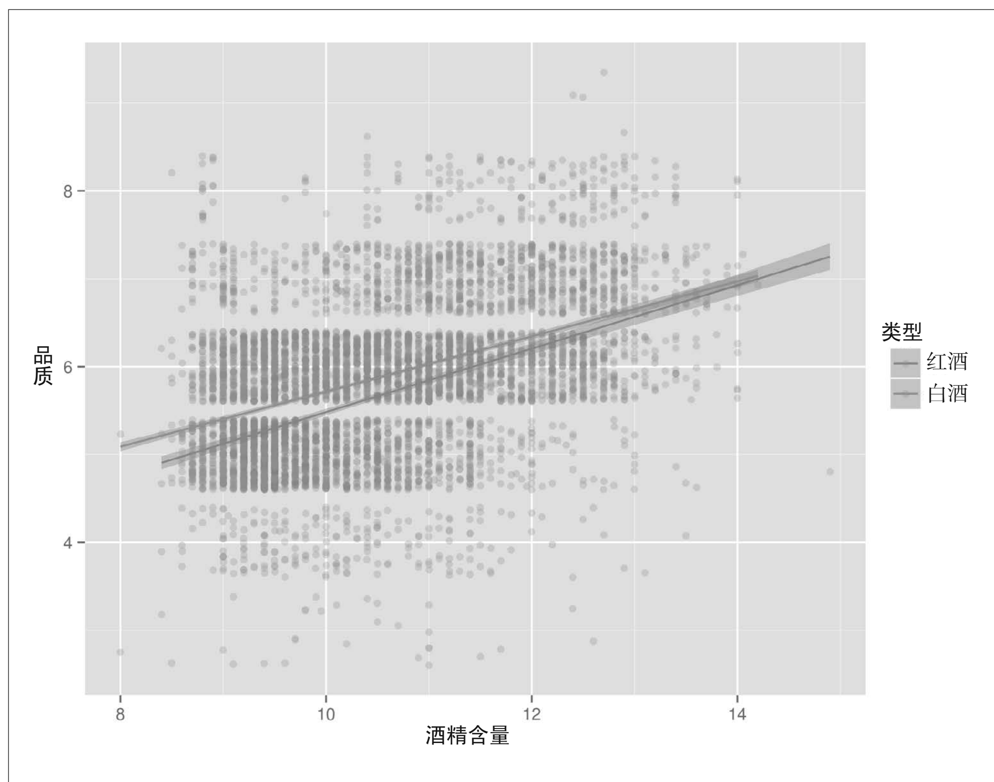


图 9-2：酒精含量和品质之间的关联

找到了！啊哈，我们进行一些建模，如何？

9.3 用Tapkee降维

降维的目的是把高维数据映射到低维空间。这里的挑战是，要在低维映射中让相似的数据保持相近。如同我们在前一节中看到的那样，我们的酒品数据集包含了 13 维特征。我们坚持使用两个维度，因为这样直观可视。

降维通常被视为探索数据步骤中的一部分。它在对特征太多的数据进行画图时很有用处。你可以做出一个散点图矩阵，但这样做每次只能显示出两个特征。降维在其他机器学习算法的预处理步骤中也是有用处的。

多数降维算法都是无监督的。这意味着它们在构建低维映射时并不使用数据标签。

本章我们将会介绍两种技术：PCA（Principal Components Analysis），它表示主成分分析（Pearson，1901）；t-SNE（t-distributed Stochastic Neighbor Embedding），它表示 t 分布随机邻近嵌入（van der Maaten & Hinton，2008）。

9.3.1 介绍Tapkee

Tapkee 是一个用于降维的 C++ 模版库（Lisitsyn，Widmer & Garcia，2013）。这个库中包括很多降维算法的实现，包括：

- 局部线性嵌入
- Isomap
- 多维标度
- PCA
- t-SNE

Tapkee 的网站（http://tapkee.lisitsyn.me/）上有更多关于这些算法的信息。尽管 Tapkee 主要是一个包含在其他应用中的库，它也提供了一个命令行工具。我们将用它在酒品数据集上进行降维。

9.3.2 安装Tapkee

如果你没有运行数据科学工具箱，就需要自行下载和编译 Tapkee。首先要确定你已经安装了 `CMake`。在 Ubuntu 里，你简单运行：

```
$ sudo apt-get install cmake
```

对于其他操作系统，请到 Tapkee 的网站上寻找操作指南。运行如下命令来下载并编译源码：

```
$ curl -sL https://github.com/lisitsyn/tapkee/archive/master.tar.gz > \
> tapkee-master.tar.gz
$ tar -xzf tapkee-master.tar.gz
$ cd tapkee-master
$ mkdir build && cd build
$ cmake ..
$ make
```

这会生成一个叫作 tapkee 的二进制可执行程序。

9.3.3 线性和非线性映射

首先，我们通过标准化把特征归一到统一的尺度下，使得每个特征具有相同的重要性。在应用机器学习算法的时候，它通常会带来更好的结果。

为了进行尺度归一，我们使用了 `Rio` 和 `cols` 的组合：

```
$ < wine-both.csv cols -C type Rio -f scale > wine-both-scaled.csv
```

现在我们应用降维技术，并用 `Rio-scatter` 将这个映射可视化显示出来（如图 9-3 所示）。

```
$ < wine-both-scaled.csv cols -C type,quality body tapkee --method pca |
> header -r x,y,type,quality | Rio-scatter x y type | display
```

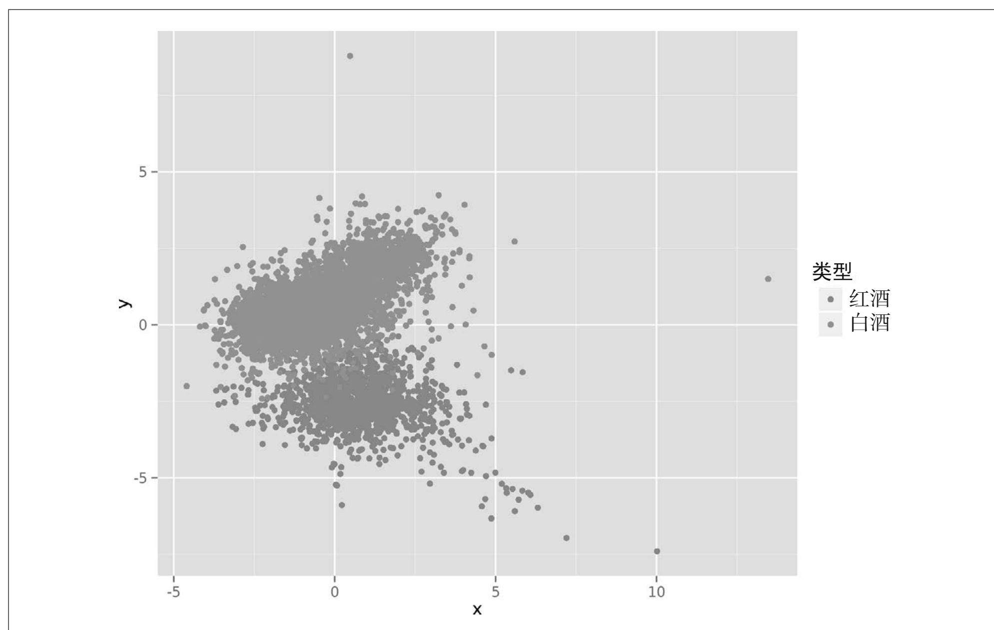


图 9-3：用 PCA 进行线性降维

```
$ < wine-both-scaled.csv cols -C type,quality body tapkee --method t-sne |
> header -r x,y,type,quality | Rio-scatter x y type | display
```

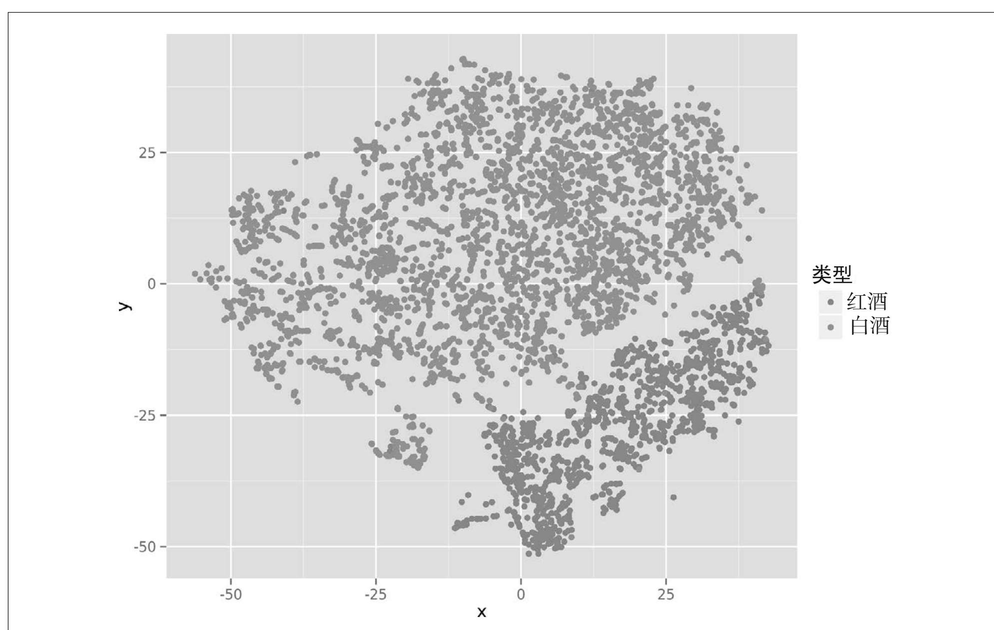


图 9-4：用 t-SNE 进行非线性降维

注意，在这两种线性方法中，并没有使用什么经典的命令行工具（也就是 GNU 的 coreutils 软件包）。现在这些都是由于创建了你自己的工具而带来的效果！

9.4 用Weka聚类

在本节中，我们会把酒品数据聚成若干组。跟降维一样，聚类通常也是非监督的。它可以用于理解数据的构成。一旦数据被聚类，你就可以根据数据点的类别进行染色，把结果可视化显示出来。对于大多数算法来说，你需要预先指定想要数据聚出几个类；而一些算法能够自动确定适合的分组数目。

对这个任务来说，我们将使用 Weka。它是由怀卡托大学的机器学习组（Hall et al.，2009）维护的。如果你已经了解 Weka，那可能知道它是一个带有用户图形界面的软件。然而，你将会看到，Weka 也可以在命令行中使用（虽然需要一些修改）。除了聚类，Weka 还可以进行分类和回归，不过我们将使用其他工具来完成这些机器学习任务。

9.4.1 介绍Weka

你可能会问：对聚类来说，一定有更好的命令行工具，不是吗？你是对的。我们要在本章里引入 Weka 的一个原因，就是想告诉你如何通过构建额外的命令行工具来克服缺陷。随着你在命令行中投入更多时间并尝试更多其他的命令行工具，有可能发生的情况是，一开始你碰见了一个非常有前景的工具，但后来发现跟你的预期不符。例如，一种常见的缺陷是，命令行工具没有正确处理标准输入或标准输出。在下一节里，我们会指出 Weka 的缺陷，并用示例说明如何克服它们。

9.4.2 在命令行里改进Weka

Weka 可以在命令行中使用，但是肯定不会很直接或者对用户友好。Weka 是用 Java 编写的，这意味着你需要运行 `java`，指定 weka.jar 文件的位置，以及指定你要调用的每个类。例如，Weka 有一个类叫作 `MexicanHat`，它会生成一个简单的数据集。要用这个类生成 10 个数据点，你可以运行：

```
$ java -cp ~/bin/weka.jar weka.datagenerators.classifiers.regression.MexicanHat\
>  -n 10 | fold
%
% Commandline
%
% weka.datagenerators.classifiers.regression.MexicanHat -r weka.datagenerators.c
lassifiers.regression.MexicanHat-S_1_-n_10_-A_1.0_-R_-10..10_-N_0.0_-V_1.0 -S 1
-n 10 -A 1.0 -R -10..10 -N 0.0 -V 1.0
%
@relation weka.datagenerators.classifiers.regression.MexicanHat-S_1_-n_10_-A_1.0
_-R_-10..10_-N_0.0_-V_1.0
```

```
@attribute x numeric
@attribute y numeric

@data

4.617564,-0.215591
-1.798384,0.541716
-5.845703,-0.072474
-3.345659,-0.060572
9.355118,0.00744
-9.877656,-0.044298
9.274096,0.016186
8.797308,0.066736
8.943898,0.051718
8.741643,0.072209
```

别担心这个命令的输出，我们一会儿就会讨论它。此刻，我们关注的是 Weka 的使用。这里有 3 点需要注意：

- 我们需要运行 `java`，这是违反直觉的。
- JAR 文件含有 2000 多个类，但其中大约只有 300 个可以直接在命令行中使用。我们怎么知道有哪些呢？
- 我们需要指定类的整个命名空间：`weka.datagenerators.classifiers.regression.MexicanHat`。我们应该怎样记住它？

这是否意味着我们将放弃使用 Weka 呢？当然不是！ Weka 包含很多有用的功能，我们下面来解决这 3 个问题。

1. 一个为Weka而改进的命令行工具

要解决第一个问题，将下列代码段保存到一个叫作 weka 的新文件里，让它可执行，并且把它移到 `PATH` 下的一个目录里：

```
#!/usr/bin/env bash
java -Xmx1024M -cp ${WEKAPATH}/weka.jar "weka.$@"
```

接着，将下面这行添加到你的 ~/.bashrc 文件里，使 `weka` 可以在任何地方调用：

```
$ export WEKAPATH=/home/vagrant/repos/weka
```

我们可以调用前面这个例子：

```
$ weka datagenerators.classifiers.regression.MexicanHat -n 10
```

现在这已经是一个改进了！

2. 可用的Weka类

如前面所提到的，文件 weka.jar 含有 2000 多个类。它们当中很多都无法直接在命令行中

使用。当我们用 -h 选项调用一个类的时候，它可以给我们一个帮助信息，我们就认为这个类在命令行中是可用的。例如：

```
$ weka datagenerators.classifiers.regression.MexicanHat -h

Data Generator options:

-h
        Prints this help.
-o <file>
        The name of the output file, otherwise the generated data is
        printed to stdout.
-r <name>
        The name of the relation.
-d
        Whether to print debug informations.
-S
        The seed for random function (default 1)
-n <num>
        The number of examples to generate (default 100)
-A <num>
        The amplitude multiplier (default 1.0).
-R <num>..<num>
        The range x is randomly drawn from (default -10.0..10.0).
-N <num>
        The noise rate (default 0.0).
-V <num>
        The noise variance (default 1.0).
```

现在这是可用的。例如，下面这个类就是不可用的：

```
$ weka filters.SimpleFilter -h
java.lang.ClassNotFoundException: -h
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:171)
        at weka.filters.Filter.main(Filter.java:1344)
-h
```

下面这个管道对 weka.jar 中的每个类都运行了带有 -h 选项的 weka，并把标准输出和标准错误保存在一个和类名同名的文件里。

```
$ unzip -l $WEKAPATH/weka.jar |
> sed -rne 's/.*(weka)\/([^g])([^$]*)\.class$/\2\3/p' |
> tr '/' '.' |
> parallel --timeout 1 -j4 -v "weka {} -h > {} 2>&1"
```

我们现在有了 749 个文件。通过如下命令，我们把不包含字符串 `Exception` 的文件名都保存到 weka.classes 里：

```
$ grep -L 'Exception' * | tee $WEKAPATH/weka.classes
```

减少到 332 个类了！这里有几个类你可能会感兴趣：

- `attributeSelection.PrincipalComponents`
- `classifiers.bayes.NaiveBayes`
- `classifiers.evaluation.ConfusionMatrix`
- `classifiers.functions.SimpleLinearRegression`
- `classifiers.meta.AdaBoostM1`
- `classifiers.trees.RandomForest`
- `clusterers.EM`
- `filters.unsupervised.attribute.Normalize`

如你所见，Weka 提供了一系列的类和函数。

3. 添加tab补全

此刻，你仍然需要自己敲入整个类名。你可以在 ~/.bashrc 文件里输出 `WEKAPATH` 之后，加入如下代码段，添加一个叫作 tab 补全的功能：

```
_completeweka() {
  local curw=${COMP_WORDS[COMP_CWORD]}
  local wordlist=$(cat $WEKAPATH/weka.classes)
  COMPREPLY=($(compgen -W '${wordlist[@]}' -- "$curw"))
  return 0
}
complete -o nospace -F _completeweka weka
```

这个函数可以利用我们之前生成的 weka.classes 文件。如果你现在在命令行中键入 `weka clu`，并按 3 次 `<Tab>` 键，你就会看到一个与聚类有关的所有类的列表：

```
$ weka clusterers.
clusterers.CheckClusterer
clusterers.CLOPE
clusterers.ClusterEvaluation
clusterers.Cobweb
clusterers.DBSCAN
clusterers.EM
clusterers.FarthestFirst
clusterers.FilteredClusterer
clusterers.forOPTICSAndDBScan.OPTICS_GUI.OPTICS_Visualizer
clusterers.HierarchicalClusterer
clusterers.MakeDensityBasedClusterer
clusterers.OPTICS
clusterers.sIB
```

```
clusterers.SimpleKMeans
clusterers.XMeans
```

生成一个命令行工具 weka，确定可用的类，并添加 tab 补全，这让 Weka 在命令行中更容易使用一点儿。

9.4.3　在CSV和ARFF格式之间转换

Weka 使用 ARFF 文件格式。它是基本 CVS 格式加上列的信息。我们会使用两种好用的命令行工具在 CSV 和 ARFF 之间转换，它们叫作 csv2arff（见示例 9-1）和 arff2csv（见示例 9-2）。

示例 9-1　将 CSV 转换为 ARFF（csv2arff）

```
#!/usr/bin/env bash
weka core.converters.CSVLoader /dev/stdin
```

示例 9-2　将 ARFF 转换为 CSV（arff2csv）

```
#!/usr/bin/env bash
weka core.converters.CSVSaver -i /dev/stdin
```

9.4.4　比较三种聚类算法

为了用 Weka 对数据聚类，我们需要另外一个定制的命令行工具来帮我们。将数据分配给已有的簇，需要用到 AddCluster 类。不幸的是，这个类无法接受标准输入的数据，甚至当我们指定 -i /dev/stdin 的时候也不行，因为它预期的是一个以 .arff 为扩展名的文件。我们觉得这是一个很烂的设计。weka-cluster 的源码是：

```
#!/usr/bin/env bash
ALGO="$@"
IN=$(mktemp --tmpdir weka-cluster-XXXXXXXX).arff

finish () {
        rm -f $IN
}
trap finish EXIT

csv2arff > $IN
weka filters.unsupervised.attribute.AddCluster -W "weka.${ALGO}" -i $IN \
-o /dev/stdout | arff2csv
```

现在我们应用 EM 聚类算法，并把簇的分配按如下方式保存：

```
$ cd data
$ < wine-both-scaled.csv csvcut -c quality,type |        ❶
> weka-cluster clusterers.EM -N 5 |                      ❷
> csvcut -c cluster > data/wine-both-cluster-em.csv      ❸
```

❶ 用统一标度的数据集，不要在聚类中使用特征 `quality` 和 `type`。

❷ 通过 `weka-cluster` 使用算法。

❸ 只保留簇的分配。

我们对 `SimpleKMeans` 和 `Cobweb` 聚类算法也运行了同样的命令。我们现在有了 3 个簇分配信息的文件。让我们创建一个 t-SNE 映射，将簇分配信息可视化出来：

```
$ < wine-both-scaled.csv csvcut -c quality,type | body tapkee --method t-sne |
> header -r x,y > wine-both-xy.csv
```

下一步，通过 `paste` 将簇的分配和 t-SNE 映射合并在一起，并用 `Rio-scatter` 生成一个散点图（见图 9-5、图 9-6 和图 9-7）：

```
$ parallel -j1 "paste -d, wine-both-xy.csv wine-both-cluster-{}.csv | "\
> "Rio-scatter x y cluster | display" ::: em simplekmeans cobweb
```

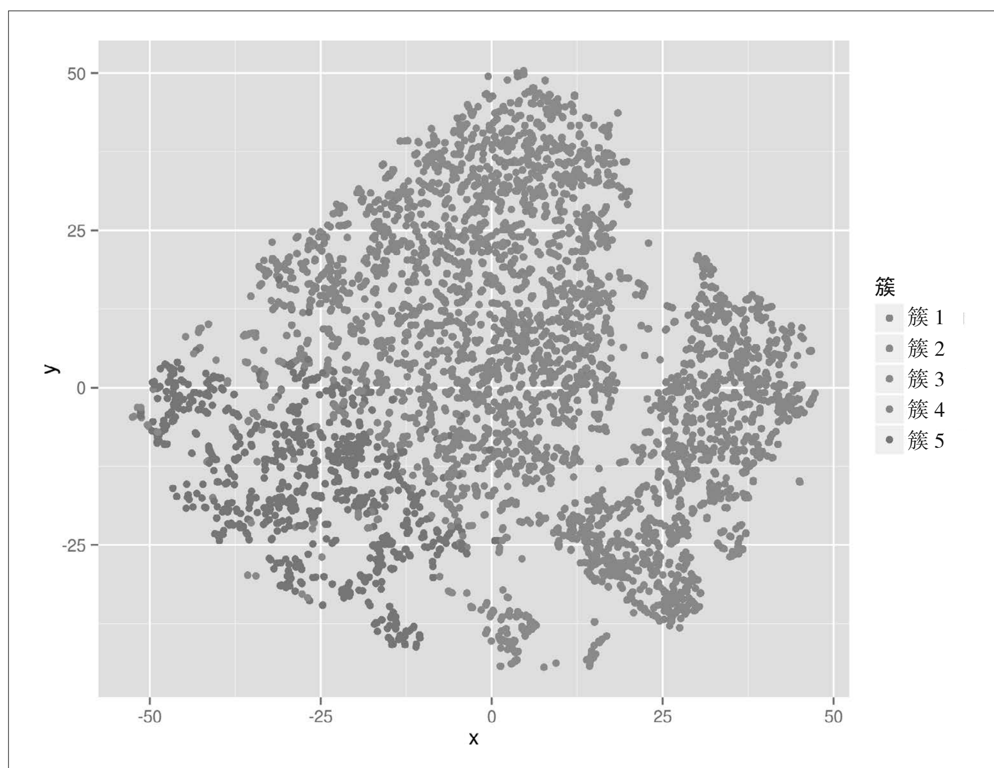


图 9-5：用 EM 算法对酒品数据聚类

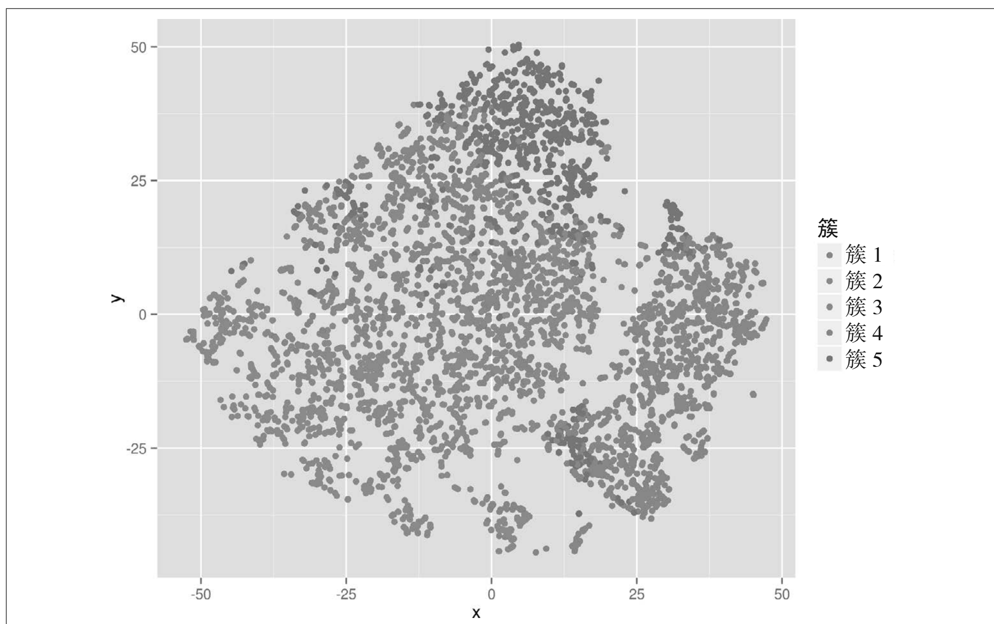


图 9-6：用 SimpleKMeans 算法对酒品数据聚类

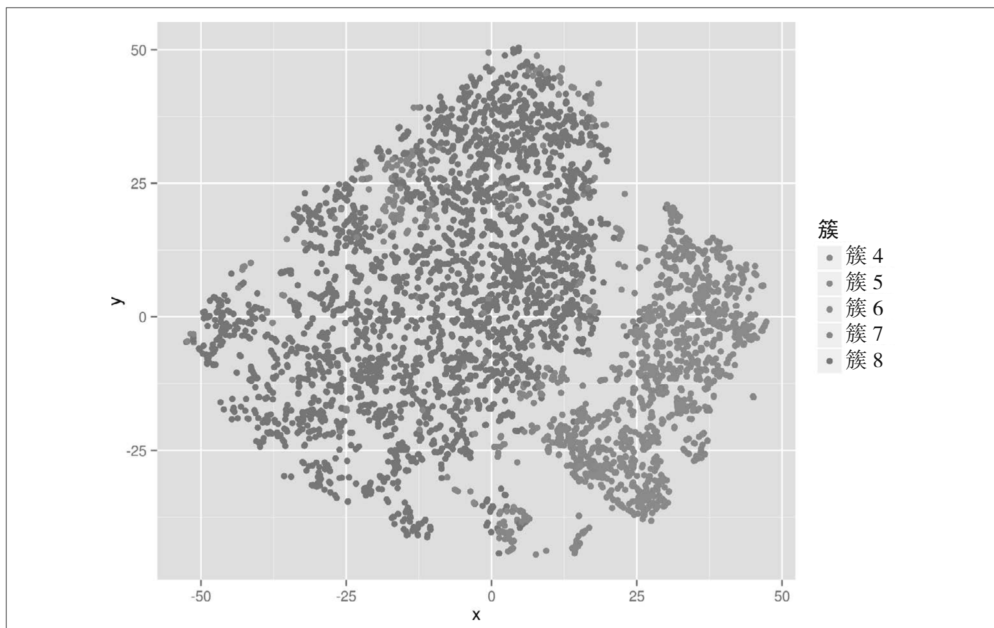


图 9-7：用 Cobweb 算法对酒品数据聚类

不可否认，我们做了好多改进 Weka 的事情。这些练习是值得的，因为也许某一天你会运行一个与你的预期不相符的命令行工具。现在你了解到，总有办法来驾驭这些命令行工具的。

9.5 通过SciKit-Learn Laboratory进行回归

在本节里，我们将会基于酒的物理化学性质来预测白酒的品质。由于品质值是一个 0 到 10 的数字，我们可以把预测品质当作一个回归任务。总的来说，我们使用训练数据用 3 个不同的算法训练了 3 个回归模型。

我们将使用 SciKit-Learn Laboratory（或者 SKLL）程序包。如果你没有使用数据科学工具箱，可以用 `pip` 安装 SKLL：

```
$ pip install skll
```

如果你正运行 Python 2.7，还需要安装如下程序包：

```
$ pip install configparser futures logutils
```

9.5.1 准备数据

SKLL 预期训练数据和测试数据具有相同的文件名，并放在独立的目录下。然而，在这个例子里，我们会使用交叉验证，这意味着我们只需要指定一个训练集。交叉验证是一个将整个数据集切分成一定数目子集的技术。这些子集叫作折。（通常使用 5 或 10 折。）

我们需要对每行添加一个标识符，以便后续很容易标识出数据点（预测值和原始数据集的顺序并不一致）：

```
$ mkdir train
$ wine-white-clean.csv nl -s, -w1 -v0 | sed '1s/0,/id,/' > train/features.csv
```

9.5.2 运行实验

创建一个叫作 predict-quality.cfg 的配置文件：

```
[General]
experiment_name = Wine
task = cross_validate

[Input]
train_location = train
featuresets = [["features.csv"]]
learners = ["LinearRegression","GradientBoostingRegressor","RandomForestRegres
sor"]
label_col = quality

[Tuning]
grid_search = false
feature_scaling = both
objective = r2
```

```
[Output]
log = output
results = output
predictions = output
```

我们用 `run_experiment` 命令行工具（Educational Testing Service，2014）运行实验：

```
$ run_experiment -l predict-quality.cfg
```

`-l` 选项指定程序以本地模式运行。SKLL 还提供了在集群中运行实验的可能性。实验运行的时间取决于所选算法的复杂度。

9.5.3 解析结果

一旦所有算法运行结束，运行结果可以在目录 output 下找到：

```
$ cd output
$ ls -1
Wine_features.csv_GradientBoostingRegressor.log
Wine_features.csv_GradientBoostingRegressor.predictions
Wine_features.csv_GradientBoostingRegressor.results
Wine_features.csv_GradientBoostingRegressor.results.json
Wine_features.csv_LinearRegression.log
Wine_features.csv_LinearRegression.predictions
Wine_features.csv_LinearRegression.results
Wine_features.csv_LinearRegression.results.json
Wine_features.csv_RandomForestRegressor.log
Wine_features.csv_RandomForestRegressor.predictions
Wine_features.csv_RandomForestRegressor.results
Wine_features.csv_RandomForestRegressor.results.json
Wine_summary.tsv
```

SKLL 为每个学习器都生成了 4 个文件：1 个是日志，2 个包含结果，还有 1 个含有预测值。此外，SKLL 生成了一个摘要文件，包含了关于每一折的很多信息（太多而无法显示在这里）。我们可以用如下 SQL 查询抽取出相关指标：

```
$ < wine_summary.tsv csvsql --query "SELECT learner_name, pearson FROM stdin "\
> "WHERE fold = 'average' ORDER BY pearson DESC" | csvlook
|----------------------------+----------------|
| learner_name               | pearson        |
|----------------------------+----------------|
| RandomForestRegressor      | 0.741860521533 |
| GradientBoostingRegressor  | 0.661957860603 |
| LinearRegression           | 0.524144785555 |
|----------------------------+----------------|
```

在这里有关联的列是 `pearson`，它表示皮尔森排序相关性。这个值在 –1 到 1 之间，它的含义是真实排序（品质分数）和预测排序之间的相关性。让我们把预测值放回到数据集里：

```
$ parallel "csvjoin -c id train/features.csv <>< output/wine_features.csv_{}"\
> ".predictions | tr '\t' ',') | csvcut -c id,quality,prediction > {}" ::: \
> RandomForestRegressor GradientBoostingRegressor LinearRegression
$ csvstack *Regres* -n learner --filenames > predictions.csv
```

并用 Rio 生成一幅图（见图 9-8）：

```
$ < predictions.csv Rio -ge 'g+geom_point(aes(quality, round(prediction), '\
> 'color=learner), position="jitter", alpha=0.1) + facet_wrap(~ learner) + '\
> 'theme(aspect.ratio=1) + xlim(3,9) + ylim(3,9) + guides(colour=FALSE) + '\
> 'geom_smooth(aes(quality, prediction), method="lm", color="black") + '\
> 'ylab("prediction")' | display
```

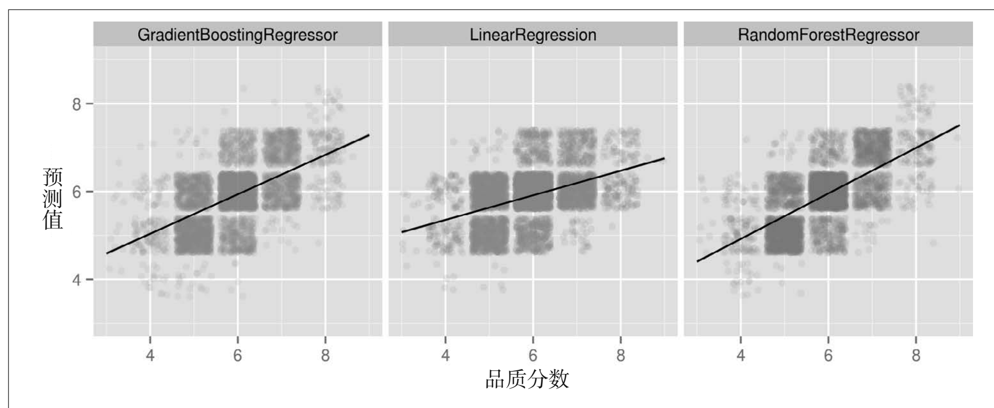


图 9-8：比较三个回归算法的输出

9.6 用BigML分类

在第四个也是最后一个建模小节中，我们将对酒是红的还是白的进行分类。对此，我们会采用一个解决方案，叫作 BigML，它提供了一个预测 API。这就是说实际的建模和预测都在云上进行。如果你需要比你的电脑更强大一些的运算能力，这是很有用处的。

尽管预测 API 相对来说还比较新，但它们正在变得越来越普及，这也是我们在本章里引入其中之一的原因。其他的预测 API 有谷歌预测 API（https://developers.google.com/prediction）和 PredictionIO（http://prediction.io/）。BigML 的一个优点在于，它提供了一个方便的命令行工具 `bigmler`（BigML，2014），可以与 API 接口。我们像使用本书中其他工具一样使用这个命令行工具，只不过在后台我们的数据会发送到 BigML 服务器，进行分类，然后把结果发送回来。

9.6.1 生成均衡的训练和测试数据集

首先，我们生成一个平衡的数据集，保证两个类被同等对待。为此，我们使用 `csvstack`（Groskop，2014）、`shuf`（Eggert，2012）、`head` 和 `csvcut`：

```
$ csvstack -n type -g red,white wine-red-clean.csv \                    ❶
> <(< wine-white-clean.csv body shuf | head -n 1600) |                  ❷
> csvcut -c fixed_acidity,volatile_acidity,citric_acid,\               ❸
> residual_sugar,chlorides,free_sulfur_dioxide,total_sulfur_dioxide,\
> density,ph,sulphates,alcohol,type > wine-balanced.csv
```

这个长长的命令分解如下：

❶ csvstack 用于融合多个数据集。它生成了一个新列 type，对于来自第一个文件 wine-red-clean.csv 的行，它的值是“red”，对于来自第二个文件的行，它的值是“white”。

❷ 通过文件重定向将第二个文件传递给 csvstack。这让我们可以用 shuf 创建一个临时文件，它会对 wine-white-clean.csv 的数据生成一个随机排列，并用 head 只选择文件头和前 1559 行。

❸ 最后，我们将数据集的列用 csvcut 记录下来，因为 bigmler 默认假设最后一列是标签。

我们使用 parallel 和 grep 计算每个类的样本个数，验证一下 wine-valanced.csv 是否真的均衡了：

```
$ parallel --tag grep -c {} wine-balanced.csv ::: red white
red      1599
white    1599
```

如你所看到的那样，数据集 wine-balanced.csv 包含有 1599 个红酒和 1599 个白酒样本。下面我们用 split（Granlund & Stallman，2012）把数据集切分成训练集和测试集：

```
$ < wine-balanced.csv header> wine-header.csv                        ❶
$ tail -n +2 wine-balanced.csv | shuf | split -d -n r/2              ❷
$ parallel --xapply "cat wine-header.csv x0{1} > wine-{2}.csv" \     ❸
> ::: 0 1 ::: train test
```

这是另一个值得分解的长命令：

❶ 用 header 获取文件头，并保存在一个叫作 wine-header.csv 的临时文件里。

❷ 用 tail 和 shuf 把红酒和白酒样本混合，然后采用 round-robin 分布把它切分到两个文件里，名为 x00 和 x01。

❸ 用 cat 把保存在 wine-header.csv 中的文件头和存储在 x00 里的行融合在一起，然后保存为 wine-train.csv，对 x01 和 wime-test.csv 也类似。--xapply 选项告诉 parallel 在两个输入源中串行循环。

再次验证一下 wine-train.csv 和 wine-test.csv 中每个类别的样本数目：

```
$ parallel --tag grep -c {2} wine-{1}.csv ::: train test ::: red white
train red       821
train white     778
test white      821
test red        778
```

我们的数据集看起来很均衡。我们现在准备好用 bigmler 调用预测 API 了。

9.6.2 调用API



你可以在 BigML 开发者页面（https://bigml.com/developers）获得一个 BigML 用户名和 API 密钥。要确保给 ~/.bashrc 中的变量 `BIGML_USERNAME` 和 `BIGML_API_KEY` 设置恰当的值。

API 调用十分直接，每个选项的含义在它的名字里显而易见：

```
$ bigmler --train data/wine-train.csv \
> --test data/wine-test-blind.csv \
> --prediction-info full \
> --prediction-header \
> --output-dir output \
> --tag wine \
> --remote
```

文件 wine-test-blind.csv 就是把 wine-test.csv 中的 `type` 列（标签）删掉。这个调用完成之后，在 output 目录中可以找到结果：

```
$ tree output
output
├── batch_prediction
├── bigmler_sessions
├── dataset
├── dataset_test
├── models
├── predictions.csv
├── source
└── source_test

0 directories, 8 files
```

9.6.3 检查结果

最感兴趣的文件是 output/predictions.csv：

```
$ csvcut output/predictions.csv -c type | head
type
white
white
red
red
white
red
red
white
red
```

我们可以拿这些预测标签和测试集中的标签进行比较。我们统计一下误分类的个数：

```
$ paste -d, <(csvcut -c type data/wine-test.csv) \         ❶
> <(csvcut -c type output/predictions.csv) |
> awk -F, '{ if ($1 != $2) {sum+=1 } } END { print sum }'   ❷
766
```

❶ 把 data/wine-test.csv 和 output/prediction.csv 中的 `type` 列合并到一起。

❷ 用 `awk` 记录两列值不同的次数。

如你所见，BigML API 在 1599 次分类中错分了 766 次。这并不是一个好结果，但注意，我们只是无目的地对一个数据集应用了一个算法，正常情况下我们不会这样做。如果在调整特征上花费更多时间，很有可能得到好得多的结果。

9.6.4　小结

BigML 的预测 API 被证明非常好用。和本书中讨论的众多命令行工具一样，我们仅仅触及了 BigML 的皮毛。你还应该注意这些额外特性：

- BigML 命令行工具 `bigmler` 还允许本地计算，这对调试十分有用。
- 结果还可以通过 BigML 的 Web 接口查看。
- BigML 还可以执行回归任务。

对于 BigML 特性的全面概述，请查看开发者页面（https://bigml.com/developers）。

尽管我们只使用了一个预测 API 进行实验，但我们相信，一般说来进行数据科学工作时预测 API 是值得考虑的。

9.7　延伸阅读

- Conway, D., & White, J. M. (2012). *Machine Learning for Hackers*. O'Reilly Media.
- Lisitsyn, S., Widmer, C., & Garcia, F. J. I. (2013). Tapkee: An Efficient Dimension Reduction Library. *Journal of Machine Learning Research*, 14, 2355–2359.
- Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling Wine Preferences by Data Mining from Physicochemical Properties. *Decision Support Systems*, 47(4), 547–553.
- Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1).
- Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11), 559–572.
- Van der Maaten, L., & Hinton, G. E. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9, 2579–2605.

第 10 章

总结

在这最后一章，本书已接近尾声。我们首先回顾一下前 9 章讨论过的内容，然后提出三条建议，并提供一些资源让你进一步探索我们触及过的相关话题。最后，如果你有任何疑问、评论或者新的命令行工具想分享，我们提供了一些联系方式。

10.1　让我们回顾一下

本书探索了用命令行完成数据科学任务的能力。一个很有趣的现象是，这个相对崭新的领域所提出的挑战，可以用这样一个经过时间考验的技术加以解决。我们希望你现在已经看到了命令行的能力。各种命令行工具提供了适用于数据科学各种任务的多种可能性。

数据科学有很多定义。第 1 章介绍了由 Mason 和 Wiggens 所定义的 OSEMN 模型，这是因为它是一个非常实际的定义，可以转化到非常具体的任务上。缩写 OSEMN 代表获取、清洗、探索、建模和解释数据。第 1 章还解释了为什么命令行非常适用于完成数据科学任务。

第 2 章解释了如何设置你自己的数据科学工具箱以及如何安装与本书相关的包。第 2 章还介绍了命令行的必要工具和概念。

有关 OSEMN 模型的章节——第 3 章（数据获取）、第 5 章（数据清洗）、第 7 章（数据探索）和第 9 章（数据建模）——致力于用命令行完成实际的任务。我们并没有用单独的一章来介绍第 5 步，即解释数据。坦白说，这是因为计算机在这里并没什么用处，更别说命令行了。不过，我们提供了一些关于这个话题的阅读资料。

在穿插其间的 3 章中，我们介绍了一些关于在命令行中进行数据科学工作的更宽泛的话

题。这些话题并不真正对应某个具体步骤。在第 4 章，我们解释了如何把单行命令和现有代码转换成可重用的命令行工具。在第 6 章，我们描述了如何用一个叫作 Drake 的命令行工具来管理数据流。在第 8 章，我们演示了如何利用 GNU Parallel 让一般的命令行工具和管道并行执行。这些话题可以应用在数据工作流中的任何一个地方。

我们不太可能把所有可用并且与数据科学相关的命令行工具都演示出来。每天都会有新的命令行工具出现。正如你到现在为止所理解的那样，本书更注重使用命令行的思想，而不是给你一个详尽的工具列表。

10.2　三条建议

你可能已经在阅读这些章节上花费了不少时间，或许还一同实践了书中的代码示例。为使你的付出获得最大的回报，并提升你继续在数据科学工作流中使用命令行的可能性，我们想给你提 3 条建议：有耐心、有所创新和肯于实践。在下面 3 个小节中我们会详细阐述这 3 条建议。

10.2.1　有耐心

我们给你的第一个建议是耐心。在命令行中从事数据工作与使用一门编程语言并不相同，因此它需要新思维。

此外，命令行工具本身并不是没有怪癖和不一致性。部分原因是它们是由很多不同的人开发而成的，并且历经了几十年时间。如果你发现自己迷失在它们令人眼花缭乱的选项中，别忘记使用 `--help`、`man`，或者你所喜欢的搜索引擎来学习更多相关知识。

尽管如此，这仍然是一个令人沮丧的体验，特别是一开始的时候。相信我们，随着你练习使用命令行及其工具，你将会变得更加精通。命令行已经有几十年的历史，并且还会存在更长时间。这是一项划算的投资。

10.2.2　有所创新

第二个相关的建议是创新。命令行是非常灵活的。结合命令行工具，你能够完成超乎你想象的事情。

我们鼓励你不要立即回到你的编程语言中。当你不得不使用编程语言的时候，想一想代码是否可以以某种方式泛化或者重用。如果可以，考虑按照第 4 章讨论的步骤，用这些代码创建你自己的命令行工具。如果你相信你的命令行工具能让其他人受益，可以更进一步，让代码开源。

10.2.3　肯于实践

第三条建议是实践。实践与创新是相关的，但仍然值得单独解释一下。在前一小节中，我们提到你不应该立即回到编程语言中。当然，命令行也有它的极限。在本书中，我们一直强调，命令行应该被当作数据科学的一个伴生方法。

我们已经讨论过在命令行中进行数据科学工作的 4 个步骤。在实践中，与步骤 4 相比，命令行更适用于步骤 1。你应该使用最适合你的任务的方法。在工作流的任何地方混合搭配方法都是完全没有问题的。命令行与其他方法、编程语言和统计环境进行整合的时候表现出色。每种方法都有一定折衷，而要想变得精通于命令行，就要学习在何时使用哪种方法。

综上所述，当你有耐心、有所创新并且肯于实践的时候，命令行将会把你变成一个更高效和高产的数据科学家。

10.3　接下来做什么

由于本书是命令行和数据科学的交叉，很多相关话题只触及了皮毛。现在，是否进一步探索这些话题取决于你。下面几个小节提供了话题列表和可供咨询的推荐资源。

10.3.1　API

- Russell, M. (2013). *Mining the Social Web* (2nd Ed.). O’Reilly Media.
- Warden, P. (2011). *Data Source Handbook*. O’Reilly Media.

10.3.2　shell编程

- Winterbottom, D. (2014). commandlinefu.com. Retrieved from http://www.commandlinefu.com.
- Peek, J., Powers, S., O’Reilly, T., & Loukides, M. (2002). *Unix Power Tools* (3rd Ed.). O’Reilly Media.
- Goyvaerts, J., & Levithan, S. (2012). *Regular Expressions Cookbook* (2nd Ed.). O’Reilly Media.
- Cooper, M. (2014). “Advanced Bash-Scripting Guide.” Retrieved May 10, 2014, from http://www.tldp.org/LDP/abs/html.
- Robbins, A., & Beebe, N. H. F. (2005). *Classic Shell Scripting*. O’Reilly Media.

10.3.3　Python、R和SQL

- Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer.
- McKinney, W. (2012). *Python for Data Analysis*. O’Reilly Media.
- Rossant, C. (2013). *Learning IPython for Interactive Computing and Data Visualization*. Packt Publishing.

10.3.4 数据解释

- Shron, M. (2014). *Thinking with Data*. O’Reilly Media.
- Patil, D. J. (2012).“Data Jujitsu”. O’Reilly Media.

10.4 联系方式

如果没有很多创造命令行及大量命令行工具的人，本书就不会写成。可以说，现在的数据科学命令行工具生态系统是社区努力的结果。我们只是略微介绍了一下很多命令行工具。每天都会有新工具问世，或许某一天你自己也会构建一个。那时，我们希望收到你的消息。当你有疑问、评论或建议的时候，请给我们写封短信，我们也会非常感激。有如下方法可以联系到我们。

- Email：jeroen@jeroenjanssens.com
- Twitter：@jeroenhjanssens
- 本书网站：http://datascienceatthecommandline.com/
- GitHub：https://github.com/jeroenjanssens/data-science-at-the-command-line

附录A

命令行工具列表

本附录将概述本书中讨论过的所有命令行工具，包括二进制可执行文件、解释型脚本，以及 Bash 内置命令和关键字。对于每个命令行工具，我们提供以下信息，如果这些信息存在并且合适的话。

- 在命令行中实际敲入的命令
- 描述
- 所属软件包的名字
- 在本书中所使用的版本
- 该版本发布的年份
- 主要作者
- 可获得更多信息的网站
- 如何安装
- 如何获得帮助
- 一个使用例子

这里所列出的所有命令行工具都包含在为本书而做的数据科学工具箱中。如何安装和使用它们，详见第 2 章。安装命令假设你所运行的系统是 Ubuntu 14.04。请注意，引用开源软件并不是很简单的事情，一些信息可能已经丢失了或者是不正确的。

alias

定义或显示别名。`alias` 是一个 Bash 内置命令。

```
$ help alias
$ alias ll='ls -alF'
```

awk

模式扫描和文本处理语言。mawk（1.3.3 版本）源自 Mike Brennan（1994）。http://invisible-island.net/mawk。

```
$ sudo apt-get install mawk
$ man awk
$ seq 5 | awk '{sum+=$1} END {print sum}'
15
```

aws

在命令行中管理 AWS 服务，例如 EC2 和 S3。AWS 命令行接口（1.3.24 版本）源自亚马逊 Web 服务（2014）。http://aws.amazon.com/cli。

```
$ sudo pip install awscli
$ aws help
$ aws ec2 describe-regions | head -n 5
{
    "Regions": [
        {
            "Endpoint": "ec2.eu-west-1.amazonaws.com",
            "RegionName": "eu-west-1"
```

Bash

GNU Bourne-Again SHell. Bash（4.3 版本）源自 Brian Fox 和 Chet Ramey（2010）。http://www.gnu.org/software/bash。

```
$ sudo apt-get install bash
$ man bash
```

bc

估算标准输入中的公式。bc（1.06.95 版本）源自 Philip A. Nelson（2006）。http://www.gnu.org/software/bc。

```
$ sudo apt-get install bc
$ man bc
$ echo 'e(1)' | bc -l
2.71828182845904523536
```

bigmler

访问 BigML 预测 API。bigmler（1.12.2 版本）源自 BigML 项目（2014）。http://bigmler.readthedocs.org。

```
$ sudo pip install bigmler
$ bigmler --help
```

body

在除第一行以外的所有行中应用一个表达式。如果你想在包含文件头的 CSV 文件中应用经典命令行工具，那这个命令是有用处的。body 源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ echo -e "value\n7\n2\n5\n3" | body sort -n
value
2
3
5
7
```

cat

连结文件和标准输入，并打印到标准输出中。cat（8.21 版本）源自 Torbjorn Granlund 和 Richard M. Stallman（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man cat
$ cat results-01 results-02 results-03 > results-all
```

cd

改变 shell 工作目录。cd 是一个 Bash 内置命令。

```
$ help cd
$ cd ~; pwd; cd ..; pwd
/home/vagrant
/home
```

chmod

改变文件模式二进制位。我们用它来让命令行工具可以执行。chmod（8.21 版本）源自 David MacKenzie 和 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man chmod
$ chmod u+x experiment.sh
```

cols

在数据列的子集中应用一个命令，并把结果合并到剩余的列中。`cols` 源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ < iris.csv cols -c species body tapkee --method pca | header -r x,y,species
```

cowsay

生成一幅牛的 ASCII 图像以及一条消息。当你构建一个特别的管道并开始感到非常沮丧的时候，这很有用。`cowsay`（3.03+dfsg1 版本）源自 Tony Monroe（1999）。

```
$ sudo apt-get install cowsay
$ man cowsay
$ echo 'The command line is awesome!' | cowsay
 ______________________________
< the command line is awesome!>
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

cp

复制文件和目录。`cp`（8.21 版本）源自 Torbjorn Granlund、David MacKenzie 和 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man cp
```

csvcut

从 CSV 数据中抽取列。和命令行工具 `cut` 一样，但用于表格数据。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ csvcut --help
```

csvgrep

过滤表格数据，使数据只剩下包含给定值或者匹配一个正则表达式的某些列。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ csvgrep --help
```

csvjoin

用一个与 SQL JOIN 操作类似的方法，将两个或者更多 CSV 表格合并到一起。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ csvjoin --help
```

csvlook

在命令行中以可读且宽度固定的格式展示一个 CSV 文件。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ csvlook --help
$ echo -e "a,b\n1,2\n3,4" | csvlook
|----+----|
| a  | b  |
|----+----|
| 1  | 2  |
| 3  | 4  |
|----+----|
```

csvsort

对 CSV 文件排序。和命令行工具 `sort` 一样，但用于表格数据。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ csvsort --help
```

csvsql

直接在 CSV 数据上执行 SQL 查询，或者将 CSV 插入到数据库中。Csvkit（0.8.0 版本）由 Christopher Groskopf（2014）创作。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ csvsql --help
```

csvstack

把多个 CSV 文件中的行堆在一起，可以有选择地对每一行增加一个分组值。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ csvstack --help
```

csvstat

打印一个 CSV 文件中所有列的描述性统计信息。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ csvstat --help
```

curl

从一个 URL 下载数据。cURL（7.35.0 版本）源自 Daniel Stenberg（2012）。http://curl.haxx.se。

```
$ sudo apt-get install curl
$ man curl
```

curlicue

为 `curl` 进行 OAuth 协议握手操作。`curlicue` 源自 Decklin Foster（2014）。https://github.com/decklin/curlicue。

```
$ git clone https://github.com/decklin/curlicue.git
```

cut

从文件的每一行中删去一块内容。`cut`（8.21 版本）源自 David M. Ihnat、David MacKenzie 和 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man cut
```

display

在任何一台 X 服务器上展示一个图片或图片序列。可以从标准输入中读取图片数据。`display`（8:6.7.7.10 版本）源自 ImageMagick Studio LLC（2009）。

```
$ sudo apt-get install imagemagick
$ man display
```

Drake

管理一个数据流。Drake（0.1.6 版本）源自 Factual（2014）。https://github.com/Factual/drake。

```
$ # Please see Chapter 6 for installation instructions.
$ drake --help
```

dseq

生成一个相对于今天的日期序列。`dseq` 源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ dseq -2 0 # day before yesterday till today
2014-07-15
2014-07-16
2014-07-17
```

echo

显示一行文本。`echo`（8.2.1 版本）源自 Brian Fox 和 Chet Ramey（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man echo
```

env

在一个修改过的环境中运行程序。通常用于指定应该由哪个解释器运行我们的脚本。`env`（8.21 版本）源自 Richard Mlynarik 和 David MacKenzie（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man env
$ #!/usr/bin/env python
```

export

为 shell 变量设置 export 属性。要让 shell 变量对其他命令行工具可用，这比较有用。export 是一个 Bash 内置命令。

```
$ help export
$ export WEKAPATH=$HOME/bin
```

feedgnuplot

在把数据传递到标准输入的时候，为 gnuplot 生成给一个脚本。feedgnuplot（1.32 版本）源自 Dima Kogan（2014）。http://search.cpan.org/perldoc?feedgnuplot.

```
$ sudo apt-get install feedgnuplot
$ man feedgnuplot
```

fieldsplit

根据某个字段值将文件分为多个文件。fieldsplit（2010-01 版本）源自 Jeremy Hinds、Jason Gessner、Jim Renwick、Norman Gocke、Rodofo Granata 和 Tobias Wolff（2010）。http://code.google.com/p/crush-tools。

```
$ # See website for installation instructions
$ fieldsplit --help
```

find

在文件层次结构中寻找文件。find（4.4.2 版本）源自 James Youngman（2008）。http://www.gnu.org/software/findutils。

```
$ sudo apt-get install findutils
$ man find
```

for

对列表中的每个元素执行命令。在第 8 章里，我们讨论了用 parallel 代替 for 的优势。for 是一个 Bash 关键字。

```
$ help for
$ for i in {A..C} "It's easy as" {1..3}; do echo $i; done
A
B
C
It's easy as
```

```
1
2
3
```

git

管理 Git 库，这是一个分布式控制系统。`git`（1:1.9.1 版本）源自 Linus Torvalds 和 Junio C. Hamano（2014）。http://git-scm.com。

```
$ sudo apt-get install git
$ man git
```

grep

打印匹配某个模式的行。`grep`（2.16 版本）源自 Jim Meyering（2012）。http:// www.gnu. org/software/grep。

```
$ sudo apt-get install grep
$ man grep
```

head

输出文件的前几行。`head`（8.2.1 版本）源自 David MacKenzie 和 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man head
$ seq 5 | head -n 3
1
2
3
```

header

添加、替换和删除文件头的行。`header` 源自 Jeroen H.M. Janssens（2014）。https://github. com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ header -h
```

in2csv

将常见但不太好的表格数据格式转化为 CSV。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ in2csv --help
```

jq

处理 JSON。`jq`（jq-1.4 版本）源自 Stephen Dolan（2014）。http://stedolan.github.com/jq。

```
$ # See website for installation instructions
$ # See website for documentation
```

json2csv

将 JSON 转换为 CSV。`json2csv`（1.1 版本）源自 Jehiah Czebotar（2014）。https://github.com/jehiah/json2csv。

```
$ go get github.com/jehiah/json2csv
$ json2csv --help
```

less

对大文件标页数。`less`（458 版本）源自 Mark Nudelman（2013）。http://www.greenwoodsoftware.com/less。

```
$ sudo apt-get install less
$ man less
$ csvlook iris.csv | less
```

ls

列出目录内容。`ls`（版本 8.2.1）源自 Richard M. Stallman 和 David MacKenzie（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man ls
```

man

阅读命令行工具的参考手册。`man`（2.6.7.1 版本）源自 John W. Eaton 和 Colin Watson（2014）。

```
$ sudo apt-get install man
$ man man
$ man grep
```

mkdir

创建目录。`mkdir`（8.21 版本）源自 David MacKenzie（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man mkdir
```

mv

移动或重命名文件或目录。`mv`（8.21 版本）源自 Mike Parker、David MacKenzie 和 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man mv
```

parallel

从标准输入中并行构建和执行 shell 命令。GNU `parallel`（20140622 版本）源自 Ole Tange（2014）。http://www.gnu.org/software/parallel。

```
$ # See website for installation instructions
$ man parallel
$ seq 3 | parallel echo Processing file {}.csv
Processing file 1.csv
Processing file 2.csv
Processing file 3.csv
```

paste

合并文件的行。`paste`（8.21 版本）源自 David M. Ihnat 和 David MacKenzie（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man paste
```

pbc

用 `parallel` 运行 `bc`。输入 CSV 的第一列映射到 `{1}`，第二列映射到 `{2}`，以此类推。`pbc` 源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ seq 5 | pbc '{1}^2'
```

```
1
4
9
16
25
```

pip

安装和管理 Python 程序包。pip（1.5.4 版本）源自 PyPA（2014）。https://pip.pypa.io。

```
$ sudo apt-get install python-pip
$ man pip
```

pwd

打印当前的工作目录名。pwd（8.21 版本）是一个 Bash 内置程序，源自 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ man pwd
$ pwd
/home/vagrant
```

Python

执行 Python，它是一个解释型、交互式的、面向对象编程语言。Python（2.7.5 版本）源自 Python Software Foundation（2014）。http://www.python.org。

```
$ sudo apt-get install python
$ man python
```

R

用 R 编程语言分析数据和生成可视化。要在 Ubuntu 上安装最新版本的 R，请遵照 http://bit.ly/ubuntu_packages_for_R 上的操作指南。R（3.1.1 版本）源自 R Foundation for Statistical Computing（2014）。http://www.r-project.org。

```
$ sudo apt-get install r-base-dev
$ man R
```

Rio

把 CSV 从标准输入中读到 R 里作为一个 data.frame，运行给定命令，并获得 CSV 或者 PNG 输出。Rio 源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-

science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ Rio -h
$ seq 10 | Rio -nf sum
55
```

Rio-scatter

用 `Rio` 从 CSV 中生成一个散点图。`Rio-scatter` 源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ < iris.csv Rio-scatter sepal_length sepal_width species > iris.png
```

rm

删除文件或目录。`rm`（8.21 版本）源自 Paul Rubin、David MacKenzie、Richard M. Stallman 和 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man rm
```

run_experiment

用 Python 程序包 scikit-learn 运行机器学习实验。SciKit-Learn Laboratory（0.26.0 版本）源自 Educational Testing Service（2014）。https://skll.readthedocs.org。

```
$ sudo pip install skll
$ run_experiment --help
```

sample

给定持续时间段和行间延迟，以一定概率打印标准输出行。`sample` 源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ sample --help
```

scp

安全地复制远程文件。`scp`（1:6.6p1 版本）源自 Timo Rinne 和 Tatu Ylonen（2014）。http://www.openssh.com。

```
$ sudo apt-get install openssh-client
$ man scp
```

scrape

用一个 XPath 查询或者 CSS3 选择器抽取 HTML 元素。`scrape` 源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-science-at-thecommand-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ curl -sL 'http://datasciencetoolbox.org' | scrape -e 'head > title'
<title>Date Science Toolbox</title>
```

sed

过滤和转换文本。`sed`（4.2.2 版本）源自 Jay Fenlason、Tom Lord、Ken Pizzini 和 Paolo Bonzini（2012）。http://www.gnu.org/software/sed。

```
$ sudo apt-get install sed
$ man sed
```

seq

打印一个数字序列。`seq`（8.21 版本）源自 Ulrich Drepper（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man seq
$ seq 5
1
2
3
4
5
```

shuf

生成随机排列。`shuf`（8.21 版本）源自 Paul Eggert（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man shuf
```

sort

对文本文件的行排序。`sort`（8.21 版本）源自 Mike Haertel 和 Paul Eggert（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man sort
```

split

把一个文件切分成若干片。`split`（8.21 版本）源自 Torbjorn Granlund 和 Richard M. Stallman（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man split
```

sql2csv

对 SQL 数据库执行任意命令，并把结果按照 CSV 格式输出。Csvkit（0.8.0 版本）源自 Christopher Groskopf（2014）。http://csvkit.readthedocs.org。

```
$ sudo pip install csvkit
$ sql2csv --help
```

ssh

登录远程主机。OpenSSH 客户端（1.8.9 版本）源自 Tatu Ylonen、Aaron Campbell、Bob Beck、Markus Friedl、Niels Provos、Theo de Raadt、Dug Song 和 Markus Friedl（2014）。http://www.openssh.com。

```
$ sudo apt-get install ssh
$ man ssh
```

sudo

以另一个用户的身份执行命令。`sudo`（1.8.9p5 版本）源自 Todd C. Miller（2013）。http://www.sudo.ws/sudo。

```
$ sudo apt-get install sudo
$ man sudo
```

tail

输出文件的最后几行。`tail`（8.21 版本）源自 Paul Rubin、David MacKenzie、Ian Lance Taylor 和 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man tail
$ seq 5 | tail -n 3
3
4
5
```

Tapkee

用各种算法对数据集降维。Tapkee 源自 Sergey Lisitsy 和 Fernando Iglesias（2014）。http://tapkee.lisitsyn.me。

```
$ # See website for installation instructions
$ tapkee --help
$ < iris.csv cols -c species body tapkee --method pca | header -r x,y,species
```

tar

生成、列举和提取 TAR 文档。`tar`（1.27.1 版本）源自 Jeff Bailey、Paul Eggert 和 Sergey Poznyakoff（2014）。http://www.gnu.org/software/tar。

```
$ sudo apt-get install tar
$ man tar
```

tee

从标准输入中读取，并写入标准输出和文件。`tee`（8.21 版本）源自 Mike Parker、Richard M. Stallman 和 David MacKenzie（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man tee
```

tr

翻译或删除字符。`tr`（8.21 版本）源自 Jim Meyering（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man tr
```

tree

以树状格式列出目录内容。tree（version 1.6.0 版本）源自 Steve Baker（2014）。https://launchpad.net/ubuntu/+source/tree。

```
$ sudo apt-get install tree
$ man tree
```

type

显示命令行工具的类型。type 是一个 Bash 内置命令。

```
$ help type
$ type cd
cd is a shell builtin
```

uniq

报告或忽略重复行。uniq（8.21 版本）源自 Richard M. Stallman 和 David MacKenzie（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man uniq
```

unpack

提取常用文件格式。unpack 源自 Patrick Brisbin（2013）。https://github.com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-commandline.git
$ unpack file.tgz
```

unrar

从 RAR 文档中提取文件。unrar（1:0.0.1+cvs20071127 版本）源自 Ben Asselstine、Christian Scheurer 和 Johannes Winkelmann（2014）。http://home.gna.org/unrar。

```
$ sudo apt-get install unrar-free
$ man unrar
```

unzip

在一个 ZIP 文档中列出、测试和提取压缩文件。unzip（6.0 版本）源自 Samuel H. Smith（2009）。

```
$ sudo apt-get install unzip
$ man unzip
```

wc

对每个文件打印空行、单词和字节个数。wc（8.21 版本）源自 Paul Rubin 和 David MacKenzie（2012）。http://www.gnu.org/software/coreutils。

```
$ sudo apt-get install coreutils
$ man wc
$ echo 'hello world' | wc -c
12
```

Weka

Weka 是一个处理数据挖掘任务的机器学习算法集合，由 Mark Hall、Eibe Frank、Geoffrey Holmes、Bernhard Pfahringer、Peter Reutemann 和 Ian H. Witten 创作。这个命令行工具允许你从命令行运行 Weka。Weka 命令行工具源自 Jeroen H.M. Janssens（2014）。https://github.com/jeroenjanssens/data-science-at-the-command-line。

```
$ git clone https://github.com/jeroenjanssens/data-science-at-the-command-line.git
```

which

查找一个命令行工具的位置。对 Bash 内置命令不管用。which 不知是由谁做的（2009）。

```
$ man which
$ which man
/usr/bin/man
```

xml2json

把 XML 转换为 JSON。Xml2Json（0.0.2 版本）源自 Francois Parmentier（2014）。https://github.com/parmentf/xml2json。

```
$ npm install xml2json-command
$ xml2json < input.xml> output.json
```

附录B

参考文献

Amazon Web Services (2014). AWS Command Line Interface Documentation. Retrieved from http://aws.amazon.com/documentation/cli/.

Conway, D., & White, J. M. (2012). *Machine Learning for Hackers*. O'Reilly Media.

Cooper, M. (2014). Advanced Bash-Scripting Guide. Retrieved May 10, 2014, from http://www.tldp.org/LDP/abs/html.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling Wine Preferences by Data Mining from Physicochemical Properties. *Decision Support Systems*, 47(4), 547–553.

Docopt. (2014). Command-line Interface Description Language. Retrieved from http://docopt.org.

Dougherty, D., & Robbins, A. (1997). *sed & awk* (2nd Ed.). O'Reilly Media.

Goyvaerts, J., & Levithan, S. (2012). *Regular Expressions Cookbook* (2nd Ed.). O'Reilly Media.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software: An Update. SIGKDD *Explorations, 11*(1).

HashiCorp. (2014). Vagrant. Retrieved May 10, 2014, from http://vagrantup.com.

Heddings, L. (2006). Keyboard Shortcuts for Bash. Retrieved May 10, 2014, from http://www.howtogeek.com/howto/ubuntu/keyboard-shortcuts-for-bash-command-shell-for-ubuntu-debian-suse-redhat-linux-etc.

Janert, P. K. (2009). *Gnuplot in Action*. Manning Publications.

Janssens, J. H. M. (2014). Data Science Toolbox. Retrieved May 10, 2014, from http://datasciencetoolbox.org.

Lisitsyn, S., Widmer, C., & Garcia, F. J. I. (2013). Tapkee: An Efficient Dimension Reduction Library. *Journal of Machine Learning Research, 14*, 2355–2359.

Mason, H., & Wiggins, C. H. (2010). A Taxonomy of Data Science. Retrieved May 10, 2014, from http://www.dataists.com/2010/09/a-taxonomy-of-data-science.

McKinney, W. (2012). *Python for Data Analysis*. O'Reilly Media.

Molinaro, A. (2005). *SQL Cookbook*. O'Reilly Media.

Oracle. (2014). VirtualBox. Retrieved May 10, 2014, from http://virtualbox.org.

Patil, D. (2012). "Data Jujitsu" . O'Reilly Media.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine, 2*(11), 559–572.

Peek, J., Powers, S., O'Reilly, T., & Loukides, M. (2002). *Unix Power Tools* (3rd Ed.). O'Reilly Media.

Perkins, J. (2010). *Python Text Processing with NLTK 2.0 Cookbook*. Packt Publishing.

Raymond, E. S. (2014). Basics of the Unix Philosophy. Retrieved from http://www.faqs.org/docs/artu/ch01s06.html.

Robbins, A., & Beebe, N. H. F. (2005). *Classic Shell Scripting*. O'Reilly Media.

Rossant, C. (2013). *Learning IPython for Interactive Computing and Data Visualization*. Packt Publishing.

Russell, M. (2013). *Mining the Social Web* (2nd Ed.). O'Reilly Media.

O'Neil, C., & Schutt, R. (2013). *Doing Data Science*. O'Reilly Media.

Shron, M. (2014). *Thinking with Data*. O'Reilly Media.

Tange, O. (2011). GNU Parallel—The Command-Line Power Tool. *;Login: The USENIX Magazine*, 36(1), 42–47. Retrieved from http://www.gnu.org/s/parallel.

Tange, O. (2014). GNU Parallel Tutorial. Retrieved from http://www.gnu.org/software/parallel/parallel_tutorial.html.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Pearson.

Van der Maaten, L., & Hinton, G. E. (2008). "Visualizing Data Using t-SNE." *Journal of Machine Learning Research*, 9, 2579–2605.

Warden, P. (2011). *Data Source Handbook*. O'Reilly Media.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer.

Wiggins, C. (2014). Public Aliases. Retrieved May 10, 2014, from https://github.com/chrishwiggins/mise/blob/master/sh/aliases-public.sh.

Wikipedia. (2014). List of HTTP status codes. Retrieved May 10, 2014, from http://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

Winterbottom, D. (2014). commandlinefu.com. Retrieved from http://www.commandlinefu.com.

Wirzenius, L. (2013). "Writing Manual Pages." Retrieved from http://liw.fi/manpages/.

作者介绍

Jeroen Janssens 是 YPlan 公司的高级数据科学家，他负责使 tonight's going out 应用的事件推荐更加个性化。Jeroen 在荷兰马斯特里赫特大学获得人工智能硕士学位，在荷兰蒂尔堡大学获得机器学习方向的博士学位。他喜欢在布鲁克林大桥上骑单车，热衷于编写工具和在 http://jeroenjanssens.com 上发表博客文章。

封面介绍

本书封面上的动物是花冠皱盔犀鸟（wreathed hornbill），拉丁文名为 Rhytidoceros undulatus。这一物种分布于东南亚大陆、印度东北部和不丹的森林。犀鸟得名于它们鸟喙上部形成的盔突。这些中空的角质结构并没有显著的用处，但可以作为种群个体之间的识别手段，用于放大叫声，或者用于性别的识别（因为雄性的盔突通常比雌性大）。花冠皱盔犀鸟与其近亲淡喉皱盔犀鸟外观非常相似，两者的区别在于前者喉囊的较低位置上有一个黑色横带。

花冠皱盔犀鸟群的规模可达 400 只，但它们是“一夫一妻制”，雌雄终生相伴。雌性鸟在雄性鸟的帮助下用粪便和泥巴封住树洞，在里面产蛋和孵化。此后四个月内，雄性鸟通过一个刚好够宽的缝隙将喙伸进树洞，喂食它的伴侣和雏鸟。雌性鸟和雏鸟离开巢穴之后，它们就从以动物猎物为食转变为以植物果实为主食。犀鸟夫妇可以在长达九年的时间里始终使用同一个巢穴。

O'Reilly 封面中的许多动物都处于濒临灭绝的境地。它们对于我们这个世界都是至关重要的。如果你想更多地了解怎样帮助它们，请浏览 http://animals.oreilly.com 网站。

封面图片来自于匈牙利雕版画。

欢迎加入

图灵社区 iTuring.cn

——最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

优惠提示：现在购买电子书，读者将获赠书款20%的社区银子，可用于兑换纸质样书。

——最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用“合集”功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

——最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、乐译、评选等多种活动，赢取积分和银子，积累个人声望。

关注图灵教育 关注图灵社区

iTuring.cn

在线出版　电子书《码农》杂志　图灵访谈 ……



QQ联系我们

图灵读者官方群I：218139230
图灵读者官方群II：164939616



微博联系我们

官方账号：@图灵教育 @图灵社区 @图灵新知
市场合作：@图灵袁野
写作本版书：@图灵小花
翻译英文书：@李松峰 @朱巍ituring @楼伟珊
翻译日文书或文章：@图灵乐馨
翻译韩文书：@图灵陈曦
电子书合作：@hi_jeanne
图灵访谈/《码农》杂志：@李盼ituring
加入我们：@王子是好人



微信联系我们



图灵教育
turingbooks



图灵访谈
ituring_interview

O'REILLY®

命令行中的数据科学

大数据时代，数据科学研究与分析日益重要。本书独树一帜，教你利用灵活的命令行工具成为高效多产的数据科学家。

为此，作者开发了数据科学工具箱，一个包含80多个命令行工具的安装简单的虚拟环境，能在Windows、OS X和Linux操作系统上运行。你将学会如何结合使用这些小而强大的命令行工具，快速地获取、清洗、探索和建模数据。

通过阅读本书，你会明白为什么命令行是一种灵活、可伸缩、易扩展的技术。即使你已经能够使用Python或R得心应手地处理数据，利用命令行也将大大改进你的数据科学工作流。

通过本书你将学会：

- **从网站、API、数据库和电子表格中获取数据**
- **对纯文本、CSV、HTML/XML和JSON格式数据进行清洗**
- **探索数据，计算描述性统计信息，进行可视化呈现**
- **管理数据科学工作流**
- **使用单行命令和已有的Python或R代码创建可重用的命令行工具**
- **实现数据密集管道的并行化和分布化**
- **使用降维、聚类、回归和分类算法对数据进行建模**

"命令行体现了Unix'简单工具'的理念，即每个工具完成一项工作，然后巧妙地连接在一起。Jeroen很专业地讨论了怎样将这个理念引入数据科学工作，展示了为什么命令行不仅能够进行简单的文件输入/输出，还是数据操作、探索，甚至建模的利器。"

——**Chris H. Wiggins**
《纽约时报》首席数据科学家，
哥伦比亚大学应用物理学
与应用数学系副教授

"这本书讲解了如何将常见的数据科学任务集成到一个连贯的工作流，涵盖了分解问题和组合方案的策略。"

——**John D. Cook**
知名应用数学、统计及
软件开发咨询顾问

Jeroen Janssens

爱思唯尔（世界领先的科技及医学出版公司）首席数据科学家，曾是纽约YPlan公司高级数据科学家。专门从事机器学习、异常检测和数据可视化。在荷兰马斯特里赫特大学获得人工智能硕士学位，在荷兰蒂尔堡大学获得机器学习博士学位。他热衷于创建数据科学的开源工具，个人网站是http://jeroenjanssens.com/。

DATA/DATA SCIENCE

封面设计：Ellie Volckhausen　张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议　计算机/大数据

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版



ISBN 978-7-115-39168-1
定价：49.00元